

Inhaltsverzeichnis

1	Das Berliner MÜ-System	1
1.1	Repräsentationsformalismen	3
1.1.1	Generalisierte Phrasenstrukturgrammatiken (GPSG)	4
1.1.2	Die konstruktive Version der GPSG	6
1.1.3	Funktor-Argument-Strukturen	7
1.1.4	Die referentielle Textrepräsentation	10
1.2	Algorithmen	13
1.2.1	Morphologische Flexion im SUTRA-System	13
1.2.2	Syntaktische Analyse mit dem Dominanz-Chart-Parser	14
1.2.3	Termersetzung	16
1.2.4	Das Anapherninterpretationsverfahren	19
1.3	Strukturübergänge	20
1.3.1	Die morphologische Analyse und Synthese	20
1.3.2	Syntaktische Analyse	20
1.3.3	Die semantische Analyse	22
1.3.4	Die konzeptuelle Analyse	23
1.3.5	Der Transfer	26
1.3.6	Die Generierung	27
2	Installationshandbuch	29
2.1	Installation des MÜ-Systems	29
2.2	Die Verzeichnisstruktur	32
2.2.1	Das MÜ-System	32
2.2.2	Die Daten für das MÜ-System	33
2.2.3	Werkzeuge zum Erstellen der linguistischen Daten	34
2.2.4	Dokumentationen	35
2.3	Prolog-Programme	35
2.3.1	Unifikation	36
2.3.2	Der Interpretierer für TES	38
2.4	Die interne Darstellung der linguistischen Daten	43
2.4.1	Komplexe Kategorien	44
2.4.2	Daten für komplexe Kategorien	46
2.4.3	Daten für Termersetzungssysteme	47
2.4.4	Daten für kontextfreie Grammatiken mit komplexen Kategorien	50
2.4.5	Daten für die morphologische Analyse bzw. Synthese	51
2.4.6	Daten für den Parser	56
2.4.7	Daten für die semantische Analyse bzw. Generierung	60
2.4.8	Daten für die konzeptuelle Analyse	62
2.4.9	Daten für das Anapherninterpretationsverfahren	63
2.4.10	Daten für den Transfer	65
2.4.11	Daten für das BACK-System	65
3	Benutzungshandbuch	67
3.1	Starten des MÜ-Systems	67
3.2	Der MÜ-System-Interpretierer	68
3.2.1	MÜ-System-Befehle	70
3.2.2	Definition von Synonymen für Prädikate	72

3.2.3	Ausschluß von Prädikaten vom Backtracking	73
3.2.4	Erzeugung komplexer Kategorien	73
3.2.5	Form der Ausgabe von Ableitungsbäumen	74
3.2.6	Datenwelten	76
3.2.7	Zähler für Ergebnisse	78
3.3	Aufruf der Übersetzungsschritte	80
3.3.1	Strukturabbildung mit dem Termersetzungsverfahren	80
3.3.2	Morphologische Analyse einer Wortform	81
3.3.3	Syntaktische Analyse eines Satzes	81
3.3.4	Semantische Analyse eines Satzes	83
3.3.5	Konzeptuelle Analyse eines Satzes	85
3.3.6	Transfer eines Satzes	88
3.3.7	Generierung eines Satzes	90
3.3.8	Morphologische Synthese einer Wortform	92
3.3.9	Übersetzung eines Satzes	92
3.4	Aktionen mit gespeicherten Repräsentationen	95
3.4.1	Ausgabe von Repräsentationen aus der Datenbasis	95
3.4.2	Löschen von Repräsentationen aus der Datenbasis	95
3.4.3	Schreiben gespeicherter Repräsentationen in eine Datei	96
3.5	Laden von linguistischen Daten	96
3.5.1	Die Datei <code>mtmenu.pl</code>	97
3.5.2	Laden sämtlicher linguistischen Daten	99
3.5.3	Laden der Daten für die syntaktische Analyse	100
3.5.4	Laden der Daten für die semantische Analyse	101
3.5.5	Laden der Daten für die konzeptuelle Analyse	102
3.5.6	Laden der Daten für die Anapherninterpretation	102
3.5.7	Laden der Daten für die Textrepräsentation	103
3.5.8	Laden der Daten für den Transfer	103
3.5.9	Laden der Daten für die Generierung	104
3.6	Veränderung von MÜ-System-Parametern	105
3.6.1	Allgemeine Parameter	106
3.6.2	Parameter für den Parser	107
3.6.3	Parameter für das TE-Verfahren	114
3.6.4	Parameter für das Anapherninterpretationsverfahren	114
A	Das Textfragment	117
B	Die Datei <code>mtmenu.pl</code>	121
C	Die Datei <code>synonyms.pl</code>	124
D	Die Datei <code>noback.pl</code>	126
	Literaturverzeichnis	128
	Verzeichnis der Abkürzungen	136
	Stichwortverzeichnis	137

1 Das Berliner MÜ-System

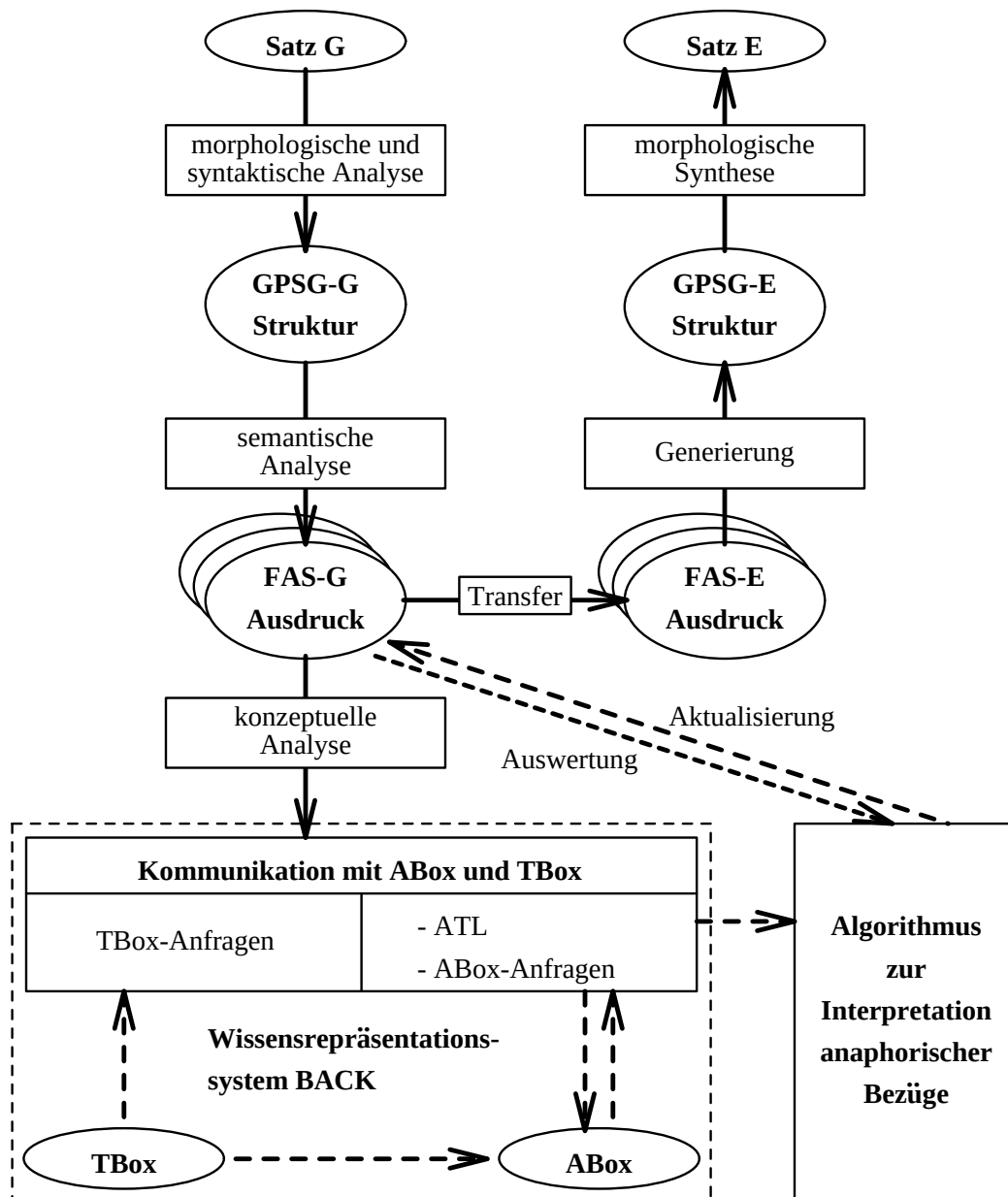
Die Architektur des MÜ-Systems im Projekts KIT-FAST, die in Abbildung 1 dargestellt ist, basiert auf einem Transfermodell, wobei die ausgangs- und zielsprachliche syntaktische Struktur durch entsprechende Generalisierte Phrasenstrukturgrammatiken (GPSG) erzeugt werden (siehe Abschnitte 1.1.1 und 1.1.2). Die ausgangs- und zielsprachliche satzsemantische Repräsentation (Funktorkomplex-Argument-Struktur, FAS) werden durch entsprechende kontextfreie Grammatiken erzeugt (siehe Abschnitt 1.1.3). Es ist nur die Übersetzungsrichtung von Deutsch nach Englisch realisiert, die erzielten Ergebnisse sind aber prinzipiell verallgemeinerbar auf andere Sprachpaare. Die satzsemantische Ebene ist die tiefste Satzrepräsentation und gleichzeitig die Transferebene, aus der die Textrepräsentation erzeugt wird.

Um einen Satz der Ausgangssprache (AS) in einen Satz der Zielsprache (ZS) zu übersetzen, müssen einige Strukturabbildungen durchgeführt werden. Die Abbildungen eines AS-Satzes in seine syntaktischen (GPSG-)Strukturen werden als syntaktische Analyse (siehe Abschnitt 1.3.2), einer AS-GPSG-Struktur in äquivalente FAS-Ausdrücke als semantische Analyse (siehe Abschnitt 1.3.3), eines AS-FAS-Ausdrucks in äquivalente ZS-FAS-Ausdrücke als Transfer (siehe Abschnitt 1.3.5), eines ZS-FAS-Ausdrucks in äquivalente ZS-GPSG-Strukturen als Generierung (siehe Abschnitt 1.3.6) und einer ZS-GPSG-Struktur in den entsprechenden ZS-Satz als morphologische Synthese bezeichnet. Alle Abbildungen, die durch einen durchgehenden Pfeil markiert sind, mit Ausnahme der syntaktischen Analyse und der morphologischen Synthese werden durch ein Verfahren auf der Basis von Termersetzung durchgeführt (siehe Abschnitt 1.2.3). Die syntaktische Analyse wird durch einen im Projekt KIT-NASEV entwickelten GPSG-Parser bewerkstelligt (siehe Abschnitt 1.2.2). Die morphologische Analyse und Synthese basieren auf dem SUTRA-System (siehe Abschnitt 1.2.1). Die Erfahrungen haben gezeigt, daß eine Zusammenfassung der syntaktischen und semantischen Analyse zu einem Termersetzungssystem sinnvoll ist. Dieses wurde aber bisher noch nicht realisiert.

Im Projekt KIT-FAST II wurden die ersten Schritte in Richtung der Lösung von textuellen Phänomenen in der MÜ gemacht. Für die Interpretation anaphorischer Bezüge des Deutschen wird eine Textrepräsentation erstellt (vgl. [Hauenschild 91], [Schmitz et al. 92], [Preuß et al. 92] und [Preuß et al. 93]), die von einer Komponente zur Anaphernresolution (siehe Abschnitt 1.2.4) verwendet wird. Das Projekt geht von einer dualen Textrepräsentation aus (vgl. [Schmitz et al. 92] und [Preuß et al. 93]). Einerseits wird der Text in seiner Linearität bzw. Verschachtelung entsprechend dem fortschreitenden Interpretationsprozeß betrachtet (strukturelle Textrepräsentation). Andererseits wird er als Ergebnis eines Interpretationsprozesses aufgefaßt und aus der Sicht der Referenzobjekte gezeigt (referentielle Textrepräsentation).

Die strukturelle Textrepräsentation wird verwendet, um einen Antezedenskandidaten nach strukturellen Kriterien zu bewerten. Dabei hat sich herausgestellt, daß die FAS die erforderlichen Informationen bereits enthält. Das Verfahren zur Interpretation anaphorischer Bezüge arbeitet deshalb auf einer Liste von FAS-Ausdrücken, so daß auch satzübergreifende Bezüge behandelt werden können.

Die referentielle Textrepräsentation wird verwendet, um die inhaltliche Konsistenz zwischen den Prädikationen über die Anapher und dem Wissen, das an die einzelnen Antezedenskandidaten geknüpft ist, zu überprüfen. Eine referentielle Textrepräsentation wird mithilfe des Wissensrepräsentationssystems BACK erstellt, das im Nachbarprojekt KIT-BACK entwickelt und in das MÜ-System des Projekts KIT-FAST integriert wurde.



Abkürzungen

G	Deutsch
E	Englisch
GPSG	Generalisierte Phrasenstrukturgrammatik
FAS	Funktor-Argument-Struktur
ATL	ABox-Tell-Language
ABox	Assertionales Wissen (referentielle Textrepräsentation)
TBox	Terminologisches Wissen (Hintergrundwissen)

Abbildung 1: Architektur des MÜ-Systems im Projekt KIT-FAST

Das BACK-System, das in [Hoppe et al. 93] und [Quantz/Kindermann 90] beschrieben wird, verwendet die KL-ONE-ähnliche formale Sprache BACK, in der terminologisches und assertionales Wissen in der TBox bzw. ABox repräsentiert werden kann. In der ABox des MÜ-Systems werden die Diskursobjekte und die Relationen zwischen ihnen repräsentiert. In der TBox werden die Selektionsrestriktionen, die von den Diskursobjekten in der ABox erfüllt werden müssen, formuliert. Langfristig sollen auch andere Formen von Hintergrundwissen wie z.B. enzyklopädisches oder übersetzungstheoretisches Wissen in der TBox modelliert werden.

ABox- und TBox-Inhalt können über eine Anfrageschnittstelle des BACK-Systems abgefragt werden. Diese Schnittstelle kann von den Abbildungsregeln (in Abbildung 1 nicht dargestellt) und dem Anapherninterpretationsverfahren benutzt werden.

Nach der morphologischen, syntaktischen und semantischen Analyse wird der Inhalt eines ausgangssprachlichen Satzes, d.h. die Diskursobjekte mit ihren Eigenschaften und Bezügen in die referentielle Textrepräsentation eingetragen. Dazu wird aus der Funktor-Argument-Struktur eines Satzes ein ATL-Ausdruck erzeugt, mit dem der Satzinhalt in die ABox des BACK-Systems eingetragen werden kann. Diese Abbildung wird als konzeptuelle Analyse (siehe Abschnitt 1.3.4) bezeichnet. Zunächst wird noch keine Information über Anaphern eingetragen. Dies erfolgt erst nach der anschließenden Interpretation ihrer Bezüge. Dabei werden die möglichen Antezedenten aus den vorangehenden Sätzen und dem aktuellen ermittelt und nach verschiedenen Kriterien bewertet. Eines der Kriterien ist die inhaltliche Konsistenz (s.o), die über ABox- und TBox-Anfragen bewertet wird. Um festzustellen, ob ein mögliches Antezedens diese Restriktion erfüllt, greift das Interpretationsverfahren über die ABox- und TBox-Anfrageschnittstelle auf die referentielle Textrepräsentation bzw. Hintergrundwissen zu. Hat das Verfahren das Antezedens ermittelt, wird die Funktor-Argument-Struktur aktualisiert, indem beim Pronomen die Referenz des ermittelten Antezedens eingetragen wird. Danach werden sämtliche Relationen, in denen das Pronomen im Satz steht in die ABox eingetragen. Die referentielle Textrepräsentation wird so inkrementell aufgebaut. Anschließend erfolgen Transfer, Generierung und morphologische Synthese.

Es reicht natürlich nicht aus, die anaphorischen Bezüge nur auf der ausgangssprachlichen Seite zu interpretieren. Sie müssen auch auf der zielsprachlichen Seite aufgelöst werden (vgl. [Schmitz et al. 92] und [Preuß et al. 93]), denn wenn beispielsweise bei der Übersetzung Verben in Funktionsverbgefüge übersetzt werden, sind in der Zielsprache andere anaphorische Bezüge möglich als in der Ausgangssprache. Diese Komponente ist jedoch zur Zeit nur rudimentär entwickelt.

1.1 Repräsentationsformalismen

Das MÜ-Modell im Projekt KIT-FAST sieht insgesamt vier Repräsentationsebenen vor. Die syntaktische und die satzsemantische Ebene sind vollständig, d.h. sowohl für die Analyse als auch für die Synthese, realisiert. Die referentielle Textrepräsentation wird nur im Rahmen der Analyse des Deutschen erstellt.

Die syntaktischen Strukturen werden von Generalisierten Phrasenstrukturgrammatiken (GPSGs) erzeugt. Dieser im zweiten Abschnitt skizzierte Grammatikformalismus gehört zu der Familie der Unifikationsgrammatiken. GPSG-Strukturen sind die Ausgangsstruktur für die semantische Analyse (siehe Abschnitt 1.3.3) und die Zielstruktur für die Generierung (siehe Abschnitt 1.3.6).

Die Funktor-Argument-Struktur (FAS), die im zweiten Abschnitt beschrieben wird,

bildet die zentrale Repräsentationsebene im MÜ-System des Projekts KIT-FAST. Diese Repräsentation wurde im Projekt entwickelt und ist die einzige Transferebene. FAS-Ausdrücke bilden die Zielstruktur der semantischen Analyse (siehe Abschnitt 1.3.3), dienen als Ein- und Ausgabestruktur für den Transfer (siehe Abschnitt 1.3.5) und sind der Ausgangspunkt für die konzeptuelle Analyse (siehe Abschnitt 1.3.4) und für die Generierung (siehe Abschnitt 1.3.6).

Die referentielle Textrepräsentation wird mithilfe des Wissensrepräsentationssystems BACK (vgl. [Hoppe et al. 93] und [Quantz/Kindermann 90]) in einer KL-ONE ähnlichen Sprache als assertionales Wissen (ABox) dargestellt. Sie wird satzweise mithilfe der *ABox-Tell-Language* (ATL) in der ABox konstruiert. Die ATL wird durch eine kontextfreie Syntax erzeugt und im letzten Abschnitt vorgestellt.

Alle drei Repräsentationsebenen lassen sich letztendlich durch kontextfreie Grammatiken mit komplexen Kategorien als Knotenmarkierungen darstellen.

1.1.1 Generalisierte Phrasenstrukturgrammatiken (GPSG)

Eine Generalisierte Phrasenstrukturgrammatik (GPSG) in ihrer ursprünglichen Form aus [Gazdar et al. 85] ist eine kontextfreie Grammatik mit einigen axiomatischen Prinzipien, welche die Zulässigkeit von Ableitungsbäumen festlegen. Im Projekt KIT-NASEV wurde daraus eine konstruktive Version entwickelt, welche die Bäume auf der Basis der axiomatischen Prinzipien konstruiert, die am Ende dieses Abschnitts vorgestellt wird. Im folgenden wird eine kurze Beschreibung der klassischen Version der GPSG gegeben.

Wie oben bereits erwähnt sind GPS-Grammatiken im Grunde genommen kontextfreie Grammatiken mit komplexen Kategorien als Knotenmarkierung. Eine komplexe Kategorie besteht aus einem Kategoriennamen (der Hauptkategorie) und einer Menge ihr zugeordneter Merkmal-Wert-Paare. Die Hauptkategorie kann aber auch als Kombination von Spezifikationen bestimmter Merkmale, z.B. *nominal* und *verbal*, aufgefaßt werden, so daß komplexe Kategorien auch nur als eine Menge von Merkmal-Wert-Paaren betrachtet werden können.

Mit der GPSG wurde erstmals die Information, die in einer kontextfreien Regel enthalten ist, aufgeteilt in zwei verschiedene Arten, nämlich die Dominanzbeziehung zwischen dem Mutterknoten und der Menge der Tochterknoten (ID-Regeln, engl. *immediate dominance*) und die Reihenfolgebeziehung zwischen den Tochterknoten (LP-Aussagen, engl. *linear precedence*). Der GPSG-Formalismus umfaßt eine sprachspezifische und eine sprachuniversale Komponente.

Die sprachspezifische Komponente

Bei der sprachspezifischen Komponente handelt es sich im wesentlichen um eine kontextfreie Grammatik im ID/LP-Format mit zusätzlichen Regeln, welche die Legalität der verwendeten komplexen Kategorien determinieren. Sie umfaßt:

- eine Menge von Immediate-Dominance-Regeln (ID-Regeln)
- eine Menge von Metaregeln, mit der bestimmte Generalisierungen über ID-Regeln ausgedrückt werden können, z.B. daß für bestimmte Aktivkonstruktionen auch entsprechende Passivkonstruktionen vorhanden sind.
- eine Menge von Linear-Precedence-Aussagen (LP-Aussagen)

- ein Lexikon
- eine Menge von Feature-Co-occurrence-Restrictions (FCRs), mit denen bestimmte Konfigurationen von Merkmalen und deren Werten in den komplexen Kategorien erlaubt werden.
- eine Menge von Feature-Specification-Defaults (FSDs), mit denen ein Wert für ein bestimmtes Merkmal determiniert wird, falls es weder durch die ID-Regeln noch durch die Merkmalinstantiierungsprinzipien (s.u.) festgelegt wird.

Die sprachuniversale Komponente

Bei der sprachuniversalen Komponente handelt es sich um Merkmalinstantiierungsprinzipien (MIPs), welche die Verteilung der Merkmalwerte in den einzelnen Teilbäumen der Tiefe 1, die als lokale Bäume bezeichnet werden, regeln. Die Menge der Merkmale enthält dabei bestimmte Klassen von Merkmalen (Foot- und Head-Merkmale). Die MIPs der klassischen Version der GPSG in [Gazdar et al. 85] sind:

- das *Foot-Feature-Prinzip* (FFP), das die Verteilung der Foot-Merkmale regelt. Das FFP fordert, daß die Spezifikation aller instantiierten, d.h. nicht in der zugrundegelegten ID-Regel festgelegten Merkmale in einem lokalen Baum identisch sind. Mit dem FFP können beispielsweise *Long-Distance*-Phänomene wie die Voranstellung einer Konstituente (Topikalisierung) oder die Verlagerung einer Konstituente an das Ende des Satzes (Extraposition) behandelt werden. Diese Konstituenten können mithilfe eines festgelegten Foot-Merkmals *slash* quasi in die entsprechende Position transportiert werden. Eine Kategorie $a(\textit{slash}: b)$ wird oft notiert als a/b und meint, daß in der Konstituente a eine Konstituente b fehlt, da sie an anderer Stelle realisiert ist.
- die *Head-Feature-Convention* (HFC), welche die Verteilung der Head-Merkmale zwischen der Mutter- und den Head-Tochter-Kategorien regelt. Mehr als eine Head-Tochter tritt normalerweise nur bei Koordination auf. Von der HFC sind nur freie Head-Merkmale betroffen, d.h. die Merkmale, deren Werte nicht durch eine ID-Regel, einer FCR, dem FFP oder dem CAP (s.u.) festgelegt sind. Die HFC fordert, daß die Spezifikation der freien Head-Merkmale jeder Head-Tochter mit denen der Mutter und die Spezifikation der freien Head-Merkmale der Mutter mit denen jeder Head-Tochter identisch sind. Um die HFC an die X-Bar-Syntax zu binden, müssen alle betroffenen Kategorien für das Merkmal *bar* (Projektionsebene einer Kategorie) spezifiziert sein.
- das *Control-Agreement-Prinzip* (CAP), das die Kongruenz zwischen zwei Konstituenten regelt, indem es die Identität der Merkmalspezifikationen fordert, in denen zwei Kategorien kongruieren müssen. Nach dem CAP können zwei Kategorien miteinander kongruieren, wenn eine (der Funktor) von der anderen (dem Argument) kontrolliert wird. Die Definition von Kontrolle hängt von den semantischen Typen der beiden Kategorien ab. Die Definition des CAP hat sich als linguistisch nicht adäquat herausgestellt, u.a. weil die Abhängigkeiten zwischen CAP und HFC zirkulär sind, da die semantischen Typen von den Head-Merkmalen *nominal* und *verbal* abhängen, und die Kontrolle auf Funktoren mit einem Argument beschränkt ist.

Aus einer GPS-Grammatik kann eine äquivalente kontextfreie Grammatik erzeugt werden, indem zunächst mithilfe der Metaregeln aus der Menge von Basis-ID-Regeln eine Menge von Objekt-ID-Regeln erzeugt wird. Diese Menge von ID-Regeln wird durch die Bildung aller durch die MIPs zugelassen Kombinationen nochmals erweitert. Aus dieser Menge werden alle ID-Regeln entfernt, die Kategorien enthalten, die nicht durch die FCRs zugelassen sind. Anschließend werden daraus kontextfreie Phrasenstrukturregeln gebildet, indem zunächst alle Reihenfolgen der Töchter in den ID-Regeln gebildet werden. Danach werden die Regeln, deren Töchter einer LP-Aussage widersprechen, aus dieser Menge entfernt.

Diese axiomatische Sicht der GPSG kann leicht zu einer kombinatorischen Explosion führen. Außerdem ist, wie oben schon angedeutet, die Reihenfolge der Anwendung der einzelnen MIPs ungeklärt. Dies führte zu der Definition einer konstruktiven Version in den Projekten KIT-NASEV und KIT-FAST.

1.1.2 Die konstruktive Version der GPSG

Die Grundidee der konstruktiven Version ist, die Menge der *Immediate-Dominance*-Regeln (ID-Regeln) nicht erst durch die Merkmalinstanziierungsprinzipien (MIPs) und Bildung aller Reihenfolgen der Töchter aufzublasen, um sie danach durch Anwendung der *Feature-Co-occurrence-Restrictions* (FCRs) und *Linear-Precedence*-Aussagen (LP-Aussagen) wieder zu reduzieren, sondern die Verteilung der Merkmalspezifikationen und die Reihenfolge der Töchter in den lokalen Bäumen so zu steuern, daß während der Analyse bzw. Synthese eines Satzes nur zulässige (lokale) Bäume konstruiert werden (vgl. [Busemann/Hauenschild 88], [Hauenschild/Busemann 88] und [Busemann 92]).

Ein erster Schritt in diese Richtung war die direkte Interpretation des ID/LP-Formats während der Analyse bzw. der Synthese. Dabei werden die Basis-ID-Regeln durch Anwendung der Metaregeln zu einer Menge von Objekt-ID-Regeln expandiert, die dann aber zusammen mit den LP-Aussagen direkt vom Parser bzw. Generator verwendet werden. Dieser Ansatz wurde in [Shieber 84], [Kilbury 84], [Dörre/Momma 85], [Weisweber 87a] und [Weisweber 88] für die Analyse und in [Busemann 92] für die Generierung entwickelt und angewendet.¹

Der Parser aus [Weisweber 88] und der Generator aus [Busemann 92] wurden im den Projekten KIT-NASEV und -FAST entwickelt und in einer früheren Version des MÜ-Systems eingesetzt. Beide Verfahren waren in der Lage, die selben linguistischen Daten zu verarbeiten und benutzten gemeinsam Komponenten für die MIPs und FCRs. In einer späteren Phase wurde der Generator durch ein Termersetzungssystem ersetzt (siehe Abschnitt 1.3.6) und der Parser erweitert. Außer der direkten Verarbeitung des ID/LP-Formats wurde er so modifiziert, daß er Metaregeln ebenfalls direkt interpretieren kann.

Um die Metaregeln direkt anwenden zu können, mußte die Terminierung der (rekursiven) Anwendung einer gegebenen Metaregelmenge garantiert sein. Dies gelingt durch die Definition einer Ordnung auf der rechten und linken Seite einer Metaregel. Diese Ordnung kann quantitativ sein, d.h. eine auf beiden Seiten der Regel vorkommende Kategorie wird rechts gelöscht, oder qualitativ, d.h. auf den Merkmalwerten der Mutterkategorie wird

¹Ein anderer Weg wurde zum Beispiel in [Evans 87], [Phillips/Thompson 87], [Briscoe et al. 87] und [Naumann 88] verfolgt. Dort sollten die hinlänglich bekannten Algorithmen, die für kontextfreie Phrasenstrukturregeln geeignet waren, verwendet werden. Deshalb wurde durch Anwendung der Metaregeln und LP-Aussagen auf die Basis-ID-Regeln in einem Vorverarbeitungsschritt eine kontextfreie Grammatik erzeugt.

eine Ordnung festgelegt, die für die gesamte Menge von Metaregeln nicht widersprüchlich sein darf. Dieser Ansatz ist in [Weisweber/Preuß 92] beschrieben.

Auf diese Weise wurden eine Reihe von Metaregeln in den Parser integriert. Für die deutsche GPSG sind dies die Subjekt-Tilgungsregel, die Slash-Terminationsregel für Topikalisierung und Extraposition, die Passivierungsregel und die Auxiliarregel, für die englische GPSG u.a. die Subjekt-Aux-Inversionsregel und die Slash-Terminationsregel.

Die FCRs und MIPs wurden im Projekt KIT-FAST gegenüber der ursprünglichen Version verändert, so daß sie sowohl in einem Vorverarbeitungsschritt als auch während der Analyse angewendet werden. In dem Vorverarbeitungsschritt werden das *Agreement-Prinzip* (AP), das aus dem *Control-Agreement-Prinzip* (CAP) hervorgegangen ist, die *Head-Feature-Convention* (HFC) und die FCRs in dieser Reihenfolge angewendet, so daß in den Basis-ID-Regeln entweder Merkmalwerte instantiiert oder variable Werte kospezifiziert werden (durch Variablenbindung in Prolog). Das *Foot-Feature-Prinzip* (FFP) und nocheinmal die FCRs werden während der Analyse angewendet. Die MIPs wurden jedoch wie folgt geändert (vgl. [Preuß 89]):

- Das FFP wurde unverändert übernommen.
- Die Anwendung des AP wird durch Spezifikationsmerkmale gesteuert. Das sind Merkmale, für deren Werte vom Grammatikschreiber eine Menge von Merkmalen festgelegt wird, deren Werte identisch sein müssen mit denen jeder anderen Kategorie im lokalen Baum, die den gleichen Wert für das Spezifikationsmerkmal hat. Die Werte für die Spezifikationsmerkmale müssen ererbt sein, d.h. schon in der ID-Regel spezifiziert sein, die dem lokalen Baum zugrundeliegt. Vom AP sind ebenfalls alle Tiefenkategorien betroffen, das sind Kategorien, die als Wert eines Merkmals im lokalen Baum vorkommen. Ist zum Beispiel der Wert des Spezifikationsmerkmals *agr* zweier Kategorien in einem lokalen Baum +, so werden die für diese Spezifikation definierten Merkmale, zu denen u.a. *person* und *numerus* gehören, bei beiden Kategorien kospezifiziert. Wenn das nicht möglich ist, wird der lokale Baum abgelehnt.
- Die HFC ist fast genauso definiert wie das AP. Der Unterschied ist, daß nur die Tiefenkategorien der Mutter berücksichtigt werden. Daß bei der HFC nur die Tiefenkategorien der Mutter betroffen sind, liegt daran, daß zum Beispiel im Deutschen auch die Head-Tochter topikalisiert werden kann, und dann als Wert des Merkmals *slash* bei der Mutter vorkommt, so daß die Mutter und ihr *slash*-Wert in den Head-Merkmalen identifiziert werden.

Die GPS-Grammatiken, die im Projekt KIT-FAST verwendet werden, sind im Abschnitt 1.3.2 beschrieben.

1.1.3 Funktor-Argument-Strukturen

Bei der Funktor-Argument-Struktur (FAS) handelt es sich formal um eine kontextfreie Syntax mit komplexen Kategorien, welche aus einer Hauptkategorie und einer Menge von Merkmal-Wert-Paaren bestehen. Dadurch daß variable, ggf. kospezifizierte Merkmalwerte angegeben werden, lassen sich linguistisch relevante Generalisierungen beschreiben. Inhaltlich wurde die FAS von drei Seiten beeinflusst (vgl. [KIT-FAST 91]):

- Funktor-Argument-Beziehungen bilden den logischen Kern der FAS. Sie sind dem Konzept der GPSG-Semantik aus [Gazdar et al. 85] ähnlich. Das Prinzip der Anwendung von Funktoren auf ein oder mehrere Argumente wurde daraus herangezogen.

- Auf der Basis des Konzepts für Funktionale Satzperspektive der Prager Schule (siehe [Sgall et al. 73]) wurde die Idee einer skalaren Ordnung der Konstituenten eines Satzes nach ihrer relativen Thematizität bzw. Rhematizität übernommen. Dieser Aspekt spielt eine wesentliche Rolle für eine adäquate Übersetzung und sollte daher beim Transfer gewahrt bleiben.
- Von EUROTRA-D wurden semantische Rollen (vgl. [Steiner et al. 88b]) und semantische Merkmale (vgl. [Zellinsky-Wibbelt 88]) übernommen, die für den lexikalischen Transfer von entscheidender Bedeutung sind.

Die Funktor-Argument-Beziehungen werden in einem lokalen FAS-Baum dadurch kodiert, daß die am weitesten links stehende Tochter als Funktor und die rechts neben ihm vorkommenden Schwestern als seine Argumente aufgefaßt werden. Die Argumente sind kanonisch geordnet. Die explizite Darstellung der kanonischen Ordnung ist entscheidend. Zum einen wird sie für die Definition von Anaphernbindung auf der Basis des o-Kommandos der HPSG verwendet. Zum anderen wird die FAS als Ein- und Ausgabe für den Transfer in der maschinellen Sprachübersetzung verwendet und im allgemeinen hängt die Wahl eines möglichen Übersetzungsäquivalents für ein Verb von den semantischen Eigenschaften seiner Argumente ab und nicht von deren Reihenfolge an der Oberfläche. Es ist in diesem Fall sinnvoll, neben den semantischen Eigenschaften der Argumente auch deren semantische Rollen in der satzsemantischen Struktur zu repräsentieren, um möglichst generelle Transferregeln formulieren zu können.

Beispielsweise sind *v-pred*, *n-pred* und *a-pred* Funktoren, die eine vordefinierte Anzahl von Termen (*term*) als Argumente haben, *det*(erminer) ein Funktor auf Termebene und *mod*(ifier) ein Funktor, der rekursiv auf einen *nom*(inalen) Komplex angewendet werden kann. Analoge Lösungen sind in logisch orientierten Semantiken allgemein bekannt. Weniger üblich ist die Repräsentation der Tempusinformation. Dafür wird eine Kategorie *fin* (für Finitkomplex) angenommen, der die Informationen über Tempus und Modus für jeden finiten Satz als Merkmalspezifikation für *pres*(ent), *fut*(ure) und *conj*(unctive) zugeordnet ist. In der FAS des Englischen wird zusätzlich noch ein Merkmal *prog*(ressive) angenommen. Durch diese Merkmale wird eine generelle Unterscheidung zwischen finiten und infiniten Sätzen (*clausefin* bzw. *clause*) erreicht, die andererseits auch viele gemeinsame Eigenschaften haben, die für die Übersetzung relevant sind, wie z.B. ihr Valenzrahmen (der bei (*um-*)zu-Infinitivkonstruktionen davon ausgeht, daß eins der Argumente an der Oberfläche nicht realisiert ist). Schließlich gibt es einen Funktor *illoc*(ution), der die Hauptsätze von eingebetteten Sätzen ohne eigene Illokution unterscheidet und dessen terminale Töchter angeben, ob es sich um einen Aussage-, Frage- oder Befehlssatz handelt.

Die thematische Struktur eines Satzes wird durch das Merkmal *them* dargestellt. Dieses Merkmal wird nur Tochterkategorien der FAS-Hauptkategorie *clause* zugeordnet, das sind im allgemeinen Kategorien mit den Hauptkategorien *v-pred* und *term*. Die Werte dieses Merkmals sind natürliche Zahlen, welche die Position der entsprechenden Konstituente in einer skalaren Ordnung der Satzoberfläche angeben. Die thematische Ordnung muß nicht unbedingt mit der kanonischen Ordnung der Argumente übereinstimmen, obwohl dies im unmarkierten Fall zutrifft.

Die semantischen Rollen der Argumente werden durch die komplementären Merkmale *conf*(iguration) (für semantische Konfiguration, entsprechend der Subkategorisierung in der Syntax) und *role* dargestellt. Konfigurationen und Rollen sind nicht notwendigerweise identisch im Original und seiner Übersetzung, obwohl das der Normalfall ist. Veränderungen der Rollenkonfiguration bei der Übersetzung sind nicht beliebig, wurden aber nicht

näher untersucht.

Die semantischen Eigenschaften (Merkmale) von Verben und Nomen werden als Werte der Merkmale *vtyp* bzw. *ntyp* in der FAS repräsentiert. Mögliche Werte für *vtyp* sind *stat(ic)*, *act(ion)* oder *ach(ievement)*, für *ntyp* *count* oder *mass*. Dabei repräsentiert *count* zählbare Nomen, bei denen ein indefiniter Artikel im Singular möglich ist, und *mass* nicht zählbare Nomen, die nicht pluralisierbar sind.

Die semantischen Eigenschaften der Adjektive werden mittels des Merkmals *grad* mit den Werten *gradable* oder *no* und des Merkmals *bound* mit den Werten *bound* oder *no* repräsentiert. Das erste Merkmal gibt an, ob das entsprechende Adjektiv mit *sehr* oder *ziemlich* modifiziert werden kann und das zweite, ob es nicht zählbare Nomen (engl. *mass nouns*) bindet oder nicht. Letzteres ist insbesondere relevant für die FAS des Englischen.

In den FAS-Ausdrücken werden die semantischen Eigenschaften der Nomen, Verben und Adjektive ausgehend vom FAS-Lexikoneintrag bis an die Konstituente, die als Argument eines Funktors auftritt (in der Regel *term*), durch Kospezifikation in den FAS-Regeln weitervererbt. Obwohl wir Redundanzen an anderen Stellen vermeiden (z.B. wird das Merkmal *num(ber)* nur am Determinator spezifiziert), scheint bei der Distribution der semantischen Eigenschaften Redundanz sehr hilfreich zu sein. Anderenfalls wäre die Formulierung von Transferregeln für Funktoren sehr schwierig, weil ihre Übersetzung normalerweise von den semantischen Eigenschaften seiner Argumente abhängt, und es wäre notwendig, den gesamten Baum bis hinunter zum terminalen Element, das auch noch rekursiv eingebettet sein kann, zu durchsuchen. Außerdem können die semantischen Eigenschaften auf dem Weg zur Kategorie *term* durch Koordination oder nicht-durchschnittsbildende Modifikation verändert werden. Dieses illustriert sehr gut das Problem, inwieweit Redundanz zugelassen werden muß, um die Formulierung von Transferregeln zu vereinfachen.

Die semantischen Eigenschaften von FAS-Konstituenten sind im Prinzip Elemente der konzeptuellen Ebene, auf der die Bedingungen für verschiedene Übersetzungsmöglichkeiten adäquater formuliert werden können, weil dort zusätzliche Informationen über den Kontext zur Verfügung stehen. Da ein Transfer aber nur auf FAS-Ausdrücken stattfindet, wurde diese Repräsentation mit diesen Informationen angereichert.

Die richtige Kombination von Präposition und Nomen wird durch die drei Merkmale *real(ization)* mit den möglichen Werten *shape* oder *no*, *ext(ension)* mit den möglichen Werten *temp(oral)* oder *no* und *act(or)* mit den möglichen Werten *actor* oder *no* sichergestellt. Dabei sind diese Merkmale für Präpositionen (Hauptkategorie *prep*) und Nomen (Hauptkategorie *noun*) definiert und in einer Präpositionalphrase identisch.

Die FAS enthält einerseits die gesamten übersetzungsrelevanten Informationen und andererseits neben der referentiellen Textrepräsentation in der ABox (siehe nächsten Abschnitt) die für die Anapherninterpretation relevanten Informationen. Das Merkmal *atyp* spezifiziert die Art, wie eine FAS-Konstituente in anaphorischen Bezügen teilnehmen kann. Der Wert *pron(oun)* bedeutet, daß die Konstituente ein Pronomen ist, *noun* ein Nomen, *relpron* ein Relativpronomen, *impl* ein implizites Subjekt und *evpron* eine Anapher, die sich auf ein Ereignis bezieht (z.B. *dafür*). Der Wert des Merkmals *atyp* wird aus dem FAS-Lexikon weitervererbt.

Das Merkmal *rtyp* gibt den Typ einer FAS-Konstituente an, auf die sich eine Anapher beziehen kann, d.h. eines der Oberkonzepte *object*, *event*, *property*, *time* oder *place*, welches das Konzept der Konstituente subsumiert. Der Wert dieses Merkmals wird ebenfalls aus dem FAS-Lexikon weitervererbt.

Weiterhin werden in der FAS die durch das Interpretationsverfahren ermittelten anaphorischen Bezüge durch Koindizierung der Werte für das Merkmal *ref(erence)* der Ana-

$\langle \text{abox-tell-expression} \rangle$	$\rightarrow \langle \text{obj-ref} \rangle :: \langle \text{atl-conc} \rangle$ $\langle \text{variable} \rangle :: \langle \text{atl-conc} \rangle$
$\langle \text{obj-ref} \rangle$	$\rightarrow \text{uc}(i)$
$\langle \text{atl-conc} \rangle$	$\rightarrow \langle \text{concept} \rangle$ $\langle \text{concept} \rangle \text{ and } \langle \text{atl-role} \rangle$
$\langle \text{atl-role} \rangle$	$\rightarrow \langle \text{role-description} \rangle : (\langle \text{abox-tell-expression} \rangle)$ $\langle \text{atl-role} \rangle \text{ and } \langle \text{atl-role} \rangle$
$\langle \text{role-description} \rangle$	$\rightarrow \text{inv } \langle \text{role} \rangle$ $\langle \text{role} \rangle$

Abbildung 2: Kontextfreie Syntax für ein Fragment der ATL

pher und des Antezedens dargestellt. Der Wert dieses Merkmals ist eine Referenz auf das entsprechende ABox-Objekt (siehe nächsten Abschnitt).

Da auch satzübergreifende anaphorische Bezüge behandelt werden, wird eine Liste von FAS-Ausdrücken als strukturelle Textrepräsentation verwendet.

Eine detaillierte Beschreibung der FAS enthält [Hauenschild/Umbach 88]. Auf ihre formale, modelltheoretische Interpretation gehen [Mahr/Umbach 90] ein. Die im Projekt KIT-FAST verwendeten Grammatiken für das Deutsche und Englische werden in [Hauenschild/Umbach 88], [Busemann/Hauenschild 89], [Busemann 92] und [KIT-FAST 91] vorgestellt.

1.1.4 Die referentielle Textrepräsentation

Im Projekt KIT-FAST wird die ABox des Wissensrepräsentationssystems BACK benutzt, um eine referentielle Textrepräsentation in einer KL-ONE-ähnlichen Sprache zu erzeugen. Diese Repräsentation wird von einer Anapherninterpretationskomponente (siehe Abschnitt 1.2.4) verwendet, um die inhaltliche Konsistenz zwischen den Prädikationen über die Anapher und dem Wissen, das an die einzelnen Antezedenskandidaten geknüpft ist, zu überprüfen (vgl. [Hauenschild 91], [Schmitz et al. 92], [Preuß et al. 92] und [Preuß et al. 93]). Die referentielle Textrepräsentation wird inkrementell erstellt, indem die Information eines jeden Satzes mithilfe der ABox-Tell-Language (ATL) als assertionales Wissen in die ABox des BACK-Systems eingetragen wird.

Die ATL wird durch eine kontextfreie Syntax generiert. In Abbildung 2 wird eine reduzierte Syntax für die ATL vorgestellt, wie sie auch im Projekt KIT-FAST in Gebrauch ist. Die vollständige Syntax ist in [Hoppe et al. 93] enthalten. Das nicht-terminale Symbol $\langle \text{concept} \rangle$ repräsentiert die Konzepte, die in einem Diskurs vorkommen und als ABox-Objekte in die ABox eingetragen werden. Das BACK-System vergibt für jedes ABox-Objekt eine eindeutige Referenz (engl. *unique constant*) mit dem Namen $\text{uc}(i)$. Das nicht-terminale Symbol $\langle \text{role} \rangle$ stellt die semantischen Rollen der Argumente eines prädikativen Konzepts dar. Wird anstelle einer ABox-Referenz $\langle \text{obj-ref} \rangle$ eine Variable ($\langle \text{variable} \rangle$) verwendet, wird ein neues Objekt mit dem entsprechenden Konzept $\langle \text{atl-conc} \rangle$ in die ABox eingetragen. Dabei instantiiert das BACK-System die Variable mit einer neuen eindeutigen ABox-Referenz.

Die ATL-Syntax kann noch erheblich vereinfacht werden, wenn man komplexe Kategorien als Knotenmarkierungen verwendet. Die Syntax in Abbildung 3, die den Kleene-

$$\begin{aligned}
\text{atl} &\rightarrow \text{equal}(\text{OBJ-REF}) \text{ with}(\text{INV}, \text{ROLE})^* \\
\text{with}(\text{INV}_1, \text{ROLE}_1) &\rightarrow \text{equal}(\text{OBJ-REF}) \text{ with}(\text{INV}_2, \text{ROLE}_2)^* \\
\text{equal}(\text{OBJ-REF}) &\rightarrow \langle \text{concept} \rangle
\end{aligned}$$

Abbildung 3: Kontextfreie Syntax für ATL-Strukturen mit komplexen Kategorien

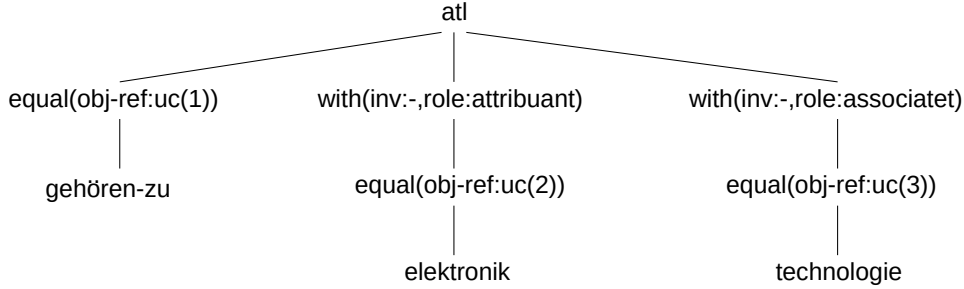


Abbildung 4: Beispiel einer ATL-Struktur des Deutschen

Operator '*' verwendet, ist äquivalent zu der in Abbildung 2. Sämtliche vordefinierte Konzepte $\langle \text{concept} \rangle$ sind in Form des letzten Regelschemas im ATL-Lexikon enthalten.

Abbildung 4 gibt ein Beispiel für eine ATL-Struktur des Deutschen, die den ATL-Ausdruck $uc(1) :: \text{gehören_zu}$ and $\text{attribuant} : (uc(2) :: \text{elektronik})$ and $\text{associated} : (uc(3) :: \text{technologie})$ nach dem Eintrag in die ABox repräsentiert. Dieser ATL-Ausdruck entspricht dem Satz *zu diesen technologien wird die elektronik gehört haben*. Allerdings sind die Informationen über das Tempus dieses Satzes nicht in der ATL-Struktur repräsentiert.

Die ATL-Struktur kann nicht direkt für den Eintrag in die ABox des BACK-Systems verwendet werden, weil die Terme, die in den TES verwendet werden, eine andere Syntax haben als die ATL-Syntax in Abbildung 2. Deshalb wird sie vorher eins-zu-eins in einen entsprechenden ATL-Ausdruck abgebildet. Abbildung 5 zeigt, wie ATL-Strukturen in ATL-Ausdrücke des BACK-Systems abgebildet werden. Dabei sind Namen, die mit einem Großbuchstaben beginnen, Variablen.

Am Ende dieses Abschnitts soll die referentielle Textrepräsentation in der ABox noch kurz vorgestellt werden (für eine ausführlichere Darstellung wird auf [Schmitz et al. 92] und [Preuß et al. 93] verwiesen). In der ABox werden die Diskursobjekte eines Textes als ABox-Objekte repräsentiert. Dazu gehören Ereignisse, Objekte, Eigenschaften und Orte, die syntaktisch durch Verben, Nomen, Adjektive bzw. lokale Präpositionen realisiert werden. Die Relationen zwischen den Diskursobjekten werden durch das nicht-terminale Symbol $\langle \text{role} \rangle$ angegeben, die in der FAS den semantischen Rollen der Argumente von Verben, Nomen und Adjektiven entsprechen. Adjektive, die ein Diskursobjekt modifizieren, stehen in einer *modifier*-Relation zu diesem Diskursobjekt. Koreferenz von Antezedens und Anapher wird in der ABox dadurch erfaßt, daß sie durch dasselbe ABox-Objekt dargestellt werden. Die Diskursobjekte in der ABox sind Instanzen der Konzepte, die in der TBox definiert sein müssen. Abbildung 6 zeigt die referentielle Textrepräsentation für den Beispieltext *Die USA ergreifen neue Initiativen. Sie erhöhen ihre Investitionen*.

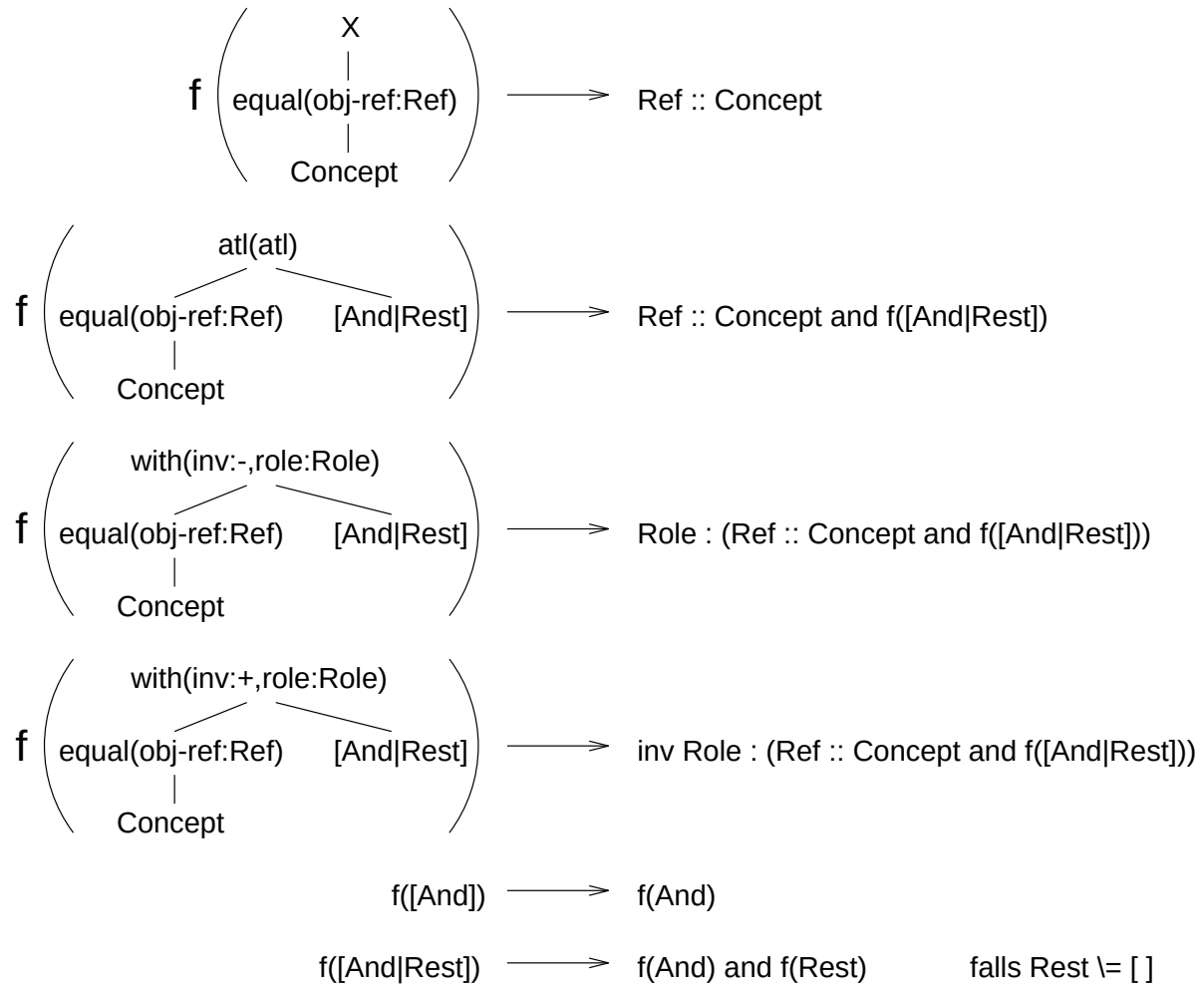


Abbildung 5: Abbildung der ATL-Strukturen in ATL-Ausdrücke des BACK-Systems

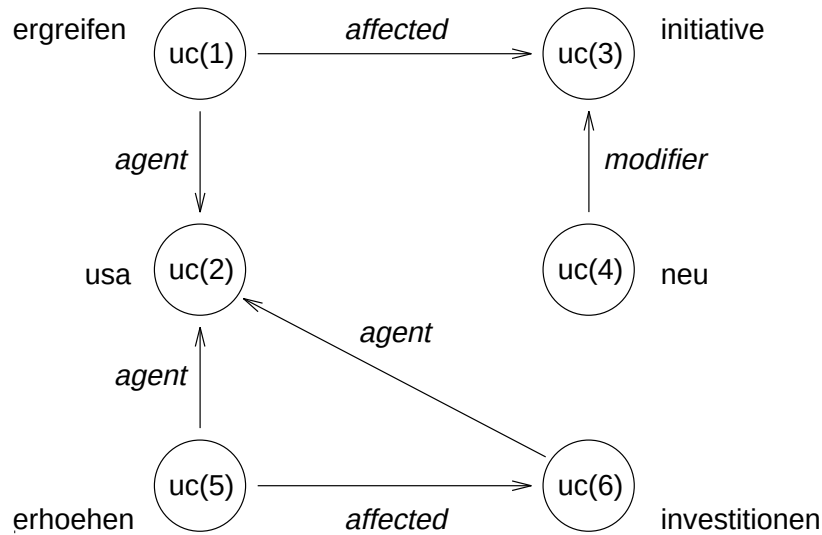


Abbildung 6: Beispiel einer referentiellen Textrepräsentation in der ABox

1.2 Algorithmen

Die einzelnen Übersetzungsschritte im MÜ-System werden durch drei Verfahren realisiert. Die morphologische Analyse und Synthese basieren auf dem SUTRA-System. Zur syntaktischen Analyse wird der Dominanz-Chart-Parser für Generalisierte Phrasenstrukturgrammatiken (GPSG, siehe Abschnitt 1.1.1 und 1.1.2) eingesetzt. Die semantische und konzeptuelle Analyse, der Transfer sowie die Generierung werden durch das Termersetzungsverfahren bewerkstelligt. Der einzige explizite Desambiguierungsschritt des MÜ-Systems findet zwischen der konzeptuellen Analyse und dem Transfer statt. Das Verfahren, das anaphorische Bezüge in deutschen Texten auflöst, wird im letzten Abschnitt beschrieben.

1.2.1 Morphologische Flexion im SUTRA-System

Das SUTRA-System (*surface transformations*, siehe [Busemann 83a] und [Busemann 83b]) wurde als Teil des Generierungsprozesses des natürlichsprachlichen Systems HAM-ANS entwickelt. Die Ausgangsstruktur enthält die Wortstämme und grammatischen Konstituenten der zu generierenden Äußerung. SUTRA bestimmt die Interpunktion sowie grammatische und stilistische Variationen der Wortstellung und flektiert die Wortstämme. Die Verarbeitung der Ausgangsstruktur erfolgt in zwei Schritten. Zunächst explizieren syntaktisch orientierte Prozesse weitere morphosyntaktische Informationen, die für den nachfolgenden Schritt benötigt werden. Außerdem werden dabei u.U. die Konstituenten des Satzes neu geordnet. Das Ergebnis ist eine lineare Kette, die im zweiten Schritt von morphologisch orientierten Prozessen verarbeitet wird, welche die Wortstämme flektiert. Im folgenden beschränke ich mich auf den zweiten Schritt, der die Grundlage für die morphologische Analyse und Synthese im MÜ-System bildet.

Sämtliche Flexionsprozesse versuchen, zunächst eine Grundform zu gewinnen, an die dann das ermittelte Flexionsmorphem angehängt wird. Im allgemeinen ist die Grundform mit dem Stamm der linearen Kette identisch, der die längste gemeinsame Teilkette aller Formen darstellt, die mit ihm gebildet werden.

Bei Personalpronomen tritt wegen ihrer Irregularität der Extremfall auf, daß die Grundform leer ist und das Morphem aus der gesamten Form besteht.

Verbstämme werden als Infinitivform mit leerem Infinitivmorphem definiert. Da keine eindeutige Grundform existiert, werden die Verben in vier Grundformen klassifiziert, die im morphologischen Lexikon eingetragen sind. Ein ebenfalls lexikalisch bestimmter Konjugationstyp legt fest, welche Formen mit welcher Grundform gebildet werden. Auf diese Weise sind fast alle irregulären Verben des Deutschen in zehn Konjugationsklassen erfaßt.

Die *Flexionsmorpheme* werden anhand der jeweils relevanten morphologischen Informationen aus einer separaten Wissensquelle gewonnen. Es werden sieben *Flexionsschemata* angenommen, die aufgrund der Wortklasse ausgewählt werden:

- Deklination von Adjektiven
- Komparation von Adjektiven
- Deklination von Artikeln und Relativpronomen
- Deklination von Quantoren und Possessivpronomen
- Deklination von Personalpronomen

- Deklination von Nomen
- Konjugation von Verben

Bei der Substantivflexion zählt zu den relevanten Informationen neben Kasus, Numerus und Genus ein Deklinationstyp, der mit dem Stamm im Lexikon assoziiert ist. Er gestattet u.a. die Berücksichtigung unterschiedlicher Pluralbildungsweisen und der Umlautung vieler Grundformen, wobei der letzte umlautbare Vokal des Stammes durch den entsprechenden Umlaut ersetzt wird (z.B. *Fall* vs. *Fälle*) bzw. der Diphthong *au* durch *äu* ersetzt wird (z.B. *Baum* vs. *Bäume*). Ein Schema von elf Deklinationstypen erfaßt nahezu alle deutschen Substantivstämme.

Bei der Deklination und Komparation von Adjektiven ist manchmal die Elision des letzten *e* im Stamm erforderlich (z.B. *teuer* vs. *teure* oder *dunkel* vs. *dunkle*). Außerdem muß bei attributivisch gebrauchten Adjektiven der jeweilige Determinationstyp der Nominalphrase berücksichtigt werden.

1.2.2 Syntaktische Analyse mit dem Dominanz-Chart-Parser

Der Parser des MÜ-Systems basiert auf einer konstruktiven Version der GPSG (siehe Abschnitt 1.1.2). Das Verfahren ist in [Weisweber 87a], [Weisweber 87b], [Weisweber 88] und [Weisweber/Preuß 92] beschrieben und baut eine Chart zur Darstellung der syntaktischen Struktur eines Eingabesatzes auf. Die ID-Regeln, LP-Aussagen, FCRs, Metaregeln und Merkmalinstanziierungsprinzipien (MIPs) der GPSG werden direkt angewendet, d.h. während der syntaktischen Analyse und nicht vorher in einem Compile-Schritt. Nur die ID-Regeln werden vorher in eine *Dominanzrelation* zerlegt, die das Kreuzprodukt aus Kategorie und Nummer einer ID-Regel ist, in der die Kategorie dominiert werden kann. Deshalb heißt der Parser auch *Dominanz-Chart-Parser (DCP)*.

Die *Kanten der Chart* sind 7-Tupel, die mithilfe des Prädikats `edge/6` und einer Referenz in die Datenbasis realisiert sind. Die Argumente des Prädikats `edge` sind

1. eine syntaktische Kategorie,
2. der Anfangspunkt der Kante in der Chart (natürliche Zahl),
3. die *Dominanzmenge* (Menge der Nummern aller ID-Regeln in der die Kategorie dominiert werden kann),
4. die Liste der Tochterbäume der entsprechenden Kategorie,
5. der FAS-Eintrag (nur für lexikalische Kategorien) und
6. die Liste der LP- und FCR-Constraints für die entsprechende Konstituente.

Der Endpunkt einer Kante wird als Referenz (Schlüssel) in die Datenbasis realisiert. Der Grund dafür ist, daß der DCP versucht, bereits analysierte Konstituenten mithilfe der Dominanzrelation *links* an eine Konstituente anzufügen, deren Kante in die Chart eingefügt wurde, um so einen lokalen Baum zu konstruieren, der zunächst nur von einer ID-Regel zugelassen wird. Die lokale syntaktische Analyse erfolgt also von rechts nach links, während die globale Analyse von links nach rechts und bottom-up erfolgt.

Der DCP geht davon aus, daß die morphologische Analyse und die Zuordnung von syntaktischen Kategorien zu den Stammformen durch einen Zugriff auf das Stammformenlexikon bereits erfolgt sind. Der Parser beginnt mit der lexikalischen Kategorie für

die erste Wortform eines Eingabesatzes, ermittelt anschließend ihre Dominanzmenge und fügt eine entsprechende Kante in die Chart ein. Danach fährt das Verfahren fort wie oben bereits beschrieben.

Könnte der DCP einen lokalen Baum konstruieren, der von einer ID-Regel zugelassen wird, werden die MIPs, FCRs, LP- und FCR-Constraints (s.u.), LP-Aussagen und Meta-regeln in der genannten Reihenfolge auf den Baum angewendet. Der Vorteil dieser späten Anwendung ist, daß zu diesem Zeitpunkt die lokalen Informationen (Merkmalspezifikationen) maximal sind. Wie ich weiter unten noch deutlich machen werde, können jedoch immer noch Fälle von nicht lokaler Information auftreten.

Alternativen treten immer dann auf, wenn einer Wortform mehrere lexikalische Kategorien zugeordnet werden können, oder wenn eine Sequenz von Konstituenten (Kanten) der Chart durch verschiedene Regeln der GPSG zugelassen wird. Der DCP konstruiert in diesem Fall sämtliche für einen Eingabesatz zulässigen syntaktischen Strukturen.²

Bei der Anwendung der LP-Aussagen und FCRs während der syntaktischen Analyse mit GPSG-Parsern kann es zu Problemen bei der Zulässigkeit von lokalen Bäumen kommen. Die Probleme entstehen, wenn durch die MIPs variable Werte für syntaktische Merkmale innerhalb eines lokalen Baumes transportieren, d.h. wenn die Merkmale in einem lokalen Baum nicht lokal spezifiziert werden. Durch die Verknüpfung von Teilbäumen, deren Wurzelknoten noch variable Merkmalwerte enthalten, zu neuen Teilbäumen können die variablen Werte u.U. im gesamten Baum verteilt werden. Wenn dann ein variabler Wert in einem lokalen Baum durch die MIPs oder FCRs instanziiert wird, so werden damit sämtliche Variablen im gesamten Baum instanziiert, die mit dem variablen Wert gebunden waren. Dies kann dazu führen, daß ein lokaler Baum oder eine Kategorie an einer völlig anderen Stelle nicht mehr LP-konsistent bzw. legal ist.

Der Grund dafür ist, daß LP-Aussagen und FCRs nur dann anwendbar sind, wenn die beteiligten Kategorien in den lokalen Strukturen Extensionen der entsprechenden Kategorien in einer LP-Aussage bzw. der Bedingung(skategorie) einer FCRs sind. Sind die Kategorien unfizierbar und stehen nicht in der o.a. Extensionsrelation, so können die entsprechenden LP-Aussagen und FCRs nicht angewendet werden.

Eine komplexe Kategorie *a* ist *Extension* einer komplexen Kategorie *b*, wenn *a* mehr Informationen enthält als *b*, d.h., wenn ein Merkmal-Wert-Paar in *b* vorkommt, so kommt es auch in *a* vor. Dies führt dazu, daß LP-Aussagen und FCRs nicht angewendet werden können, wenn die Kategorien eines lokalen Baumes zwar mit den Kategorien in den LP-Aussagen bzw. FCRs unfizierbar sind, d.h. die Informationen in den Kategorien widersprechen sich nicht, aber die Extensionsrelation ist zwischen ihnen nicht gegeben (vgl. [Weisweber 88]). Dies ist immer dann der Fall, wenn bestimmte Merkmalwerte in Kategorien, die in den Strukturen vorkommen, bei der Anwendung noch nicht bekannt, d.h. variabel sind. Ist das bei der Anwendung einer LP-Aussage oder FCR der Fall, so wird für jede LP-Aussage oder FCR eine Disjunktion von *Constraints* der Form *eq*(*Merkmal*, *Wert1*, *Wert2*) oder *neq*(*Merkmal*, *Wert1*, *Wert2*) ermittelt. *Wert1* ist der Wert des Merkmals in einer lokalen Struktur (Variable) und *Wert2* der Wert in der LP-Aussage oder FCR. Die Disjunktionen von Constraints für LP-Aussagen oder FCRs werden durch Konjunktion (logisch *und*) zusammengefaßt, so daß eine Art konjunktive Normalform

²Die Ausgabe des Parsers sind entsprechende GPSG-Ableitungsbäume, während sie zum Zweck der Weiterverarbeitung als GPSG-Terme abgespeichert werden, in denen die Wortformen und lexikalischen Kategorien bereits durch ihre FAS-Entsprechungen ersetzt sind. GPSG-Kategorien-Terme unterscheiden sich von GPSG-Kategorien im wesentlichen durch das zusätzliche erste Argument, welches die Zugehörigkeit einer Kategorie zum (syntaktischen) Vokabular markiert (siehe Abschnitt 2.4.1).

für alle Constraints entsteht. Die Constraints machen Aussagen darüber, welche Werte bestimmte Merkmale in einer lokalen Struktur bekommen dürfen, damit diese Struktur nach den LP-Aussagen und FCRs zulässig ist. Gilt in einer Disjunktion für ein Constraint $\text{eq}(M, W1, W2)$ mit $W1 = W2$ oder $\text{neq}(M, W1, W2)$ mit $W1 \neq W2$, so ist dieses Constraint wahr und damit ist die gesamte Disjunktion von Constraints wahr, die deshalb aus der konjunktiven Normalform entfernt werden kann. Gilt für sämtliche Constraints $\text{eq}(M, W1, W2)$ oder $\text{neq}(M, W1, W2)$ in einer Disjunktion $W1 \neq W2$ bzw. $W1 = W2$, so sind sämtliche Constraints falsch und damit ist die gesamte Disjunktion von Constraints falsch. Das hat zur Folge, daß die konjunktive Normalform ebenfalls falsch ist und die entsprechende lokale Struktur abgelehnt werden muß, weil entweder die lokale Struktur oder einer ihrer Unterstrukturen einer LP-Aussage oder FCR widerspricht. Diese Auswertung der Constraints führt entweder zu einem Widerspruch (die konjunktive Normalform ist falsch), zu einer leeren Menge von Constraints (die konjunktive Normalform ist wahr) oder einer nicht weiter auswertbaren konjunktive Normalform, in denen die Werte aus den Strukturen noch nicht feststehen. Im letzteren Fall wird die konjunktive Normalform im Baum an den nächst höheren Knoten weitergereicht.

Eine Disjunktion von Constraints und damit die entsprechende lokale Struktur wird abgelehnt, wenn sämtliche Constraints der Disjunktion falsch sind. Beispielsweise wird die Disjunktion $[\text{neq}(mc, +, +), \text{eq}(ac, -, +)]$ abgelehnt, weil die Aussage $+$ $\neq$ $+$ oder $-$ $=$ $+$ immer falsch ist. Ein Teilbaum mit den entsprechenden Constraints an der Wurzel würde aus diesem Grund abgelehnt.

1.2.3 Termersetzung

In der Regel wird die Termersetzung (TE) zur Lösung von Gleichheitsproblemen in Deduktionssystemen eingesetzt, mit dem Ziel, zu beweisen, ob eine Gleichung aus einer Menge von Axiomgleichungen folgt. Im Berliner MÜ-System wurde sie jedoch für die Verarbeitung natürlicher Sprache nutzbar gemacht und wird für Strukturabbildungen, d.h. für die semantische und konzeptuelle Analyse, den Transfer und die Generierung eingesetzt (siehe Abschnitt 1).

Das Grundprinzip der TE ist, Gleiches durch Gleiches zu ersetzen. Dabei werden sukzessive in den Termen auf beiden Seiten der Gleichung Unterterme durch äquivalente Terme gemäß der Axiomgleichungen ersetzt, bis ein äquivalenter minimaler³ Term auf beiden Seiten erreicht ist. Sind diese literal identisch, folgt die Gleichung aus den Axiomen. Eine gute Einführung in die Theorie der Termersetzungssysteme geben [Bläsius/Bürckert 87] und [Huet/Oppen 80].

Ein Ersetzungs- oder auch Reduktionssystem besteht aus einer Menge von Ersetzungsregeln, die eine binäre irreflexive Relation ' $\rightarrow$ ' zwischen syntaktischen Objekten definieren. Wenn die Objekte Terme erster Ordnung sind, handelt es sich um ein Termersetzungssystem (TES). Terme erster Ordnung können aus Signaturen für Termalgebren erzeugt werden (vgl. [Ehrig/Mahr 85] und [Ehrig/Mahr 89]). Die Signaturen können durch Gleichungsaxiome zu einer Spezifikation erweitert werden. Die Axiome definieren Äquivalenzklassen von Termen. Die Gleichungsmengen können mit dem Knuth-Bendix-Algorithmus in ein kanonisches TES umgewandelt werden (vgl. [Knuth/Bendix 70]).⁴

³*Minimal* wird hier nicht nur im quantitativen Sinn verstanden. In der Termersetzung, insbesondere bei der Verwendung zur Verarbeitung natürlicher Sprache, ist es sinnvoll, *minimal* auch qualitativ zu interpretieren. Minimale Terme sind nicht weiter reduzierbar.

⁴Ein kanonisches TES ist terminierend und konfluent. Letzteres heißt, daß die Reihenfolge der Anwen-

Eine Termersetzungsregel (TE-Regel) ist eine gerichtete Gleichung $\lambda \rightarrow \rho$ und besteht aus einer linken Seite λ und einer rechten Seite ρ . Sie definiert, daß der Term λ zum Term ρ reduziert werden kann. Man sagt auch ρ ist aus λ ableitbar. Wenn beide Seiten einer TE-Regel Terme ohne Variablenvorkommen (Grundterme) sind, definiert sie die Relation ' $\rightarrow$ ' für genau ein Paar von Termen. Kommen in den Termen jedoch Variablen vor (offene Terme), so definiert diese TE-Regel die Relation ' $\rightarrow$ ' für Klassen von Termpaaren. Durch Anwendung der TE-Regeln eines TES kann aus einem Grundterm ein äquivalenter Term abgeleitet werden.

Termersetzungssysteme sind bestimmte Ausprägungen von Produktionssystemen, die aus einer Regelbasis, einer Datenbasis und einem Interpretierer bestehen (vgl. [Davis/King 77]). Die Regelbasis enthält eine Menge von Produktionen. Die Datenbasis repräsentiert den aktuellen Zustand des Systems und der Interpretierer steuert die Anwendung der Produktionen. Eine Produktion besteht aus einem Bedingungs- und einem Aktionsteil. Ist der Bedingungsteil einer Produktion durch die Datenbasis erfüllt, so ist sie anwendbar, d.h. der Aktionsteil wird durch den Interpretierer ausgeführt und die Datenbasis entsprechend verändert.

Der Interpretierer steuert die Auswahl der anzuwendenden Produktionen. Dabei können Konflikte entstehen, die durch bestimmte Strategien gelöst werden müssen (vgl. [Davis/King 77]):

- Die Produktionen sind total geordnet und die anwendbare Produktion mit der höchsten Priorität wird angewendet (Regelordnung).
- Die Elemente in der Datenbasis sind total geordnet und die Produktion, die auf das Element mit der höchsten Priorität anwendbar ist, wird angewendet (Datenordnung).
- Die spezifischste Produktion wird zuerst angewendet (Generalitätsordnung).
- Die Anwendung der Produktionen wird durch ein Präzedenz-Netz, das Zyklen enthalten kann, gesteuert (Regelpräzedenz).
- Von den anwendbaren Produktionen wird die zuletzt angewendete oder diejenige, die auf das zuletzt bearbeitete Element der Datenbasis anwendbar ist, angewendet (Aktualitätsordnung).

Die meisten Interpretierer verwenden einen Steuergraphen, der auch Zyklen enthalten kann und manuell spezifiziert werden muß. Er legt fest, in welcher Reihenfolge und wie die Produktionen angewendet werden. Diese Reihenfolge muß nicht unbedingt mit der inhärenten Ordnung der Produktionen übereinstimmen.⁵ Dies kann ausgenutzt werden, um die Kombinatorik bei den Anwendungsmöglichkeiten von Produktionen zu beschränken, so daß überflüssige Ableitungen und Mehrfachableitungen verhindert werden können. Dies ist möglich, wenn die explizite Ordnung der Produktionen nicht mit der inhärenten übereinstimmt. Allerdings hat die explizite Ordnung den Nachteil, daß der Produktionenschreiber einen vollständigen Überblick über das gesamte, meist sehr komplexe Produktionssystem haben muß.

dung, d.h. der Ableitungsweg beliebig ist, da immer genau ein und derselbe minimale Term das Ergebnis einer Ableitung ist.

⁵Mit *inhärenter Ordnung* ist die Ordnung der Produktionen gemeint, die zur Erzeugung aller gemäß der Produktion zulässigen Lösungen führen, d.h. zu einer vollständigen Interpretation der Produktionen.

In der TE entspricht ein vorgegebener Grundterm, aus dem mithilfe der TE-Regeln ein äquivalenter Term abgeleitet werden soll, der Datenbasis. Die TE-Regeln entsprechen den Produktionen. Die linke Seite einer TE-Regel ist die strukturelle Bedingung für deren Anwendung, die erfüllt ist, wenn sie mit einem Unterterm des Grundterms unifizierbar ist. Wenn das der Fall ist, wird der Unterterm durch die rechte Seite der TE-Regel ersetzt.

Aus fast allen TES, die für die Verwendung in Deduktionssystemen untersucht werden, lassen sich kanonische, d.h. terminierende und konfluente TES erzeugen. Ergebnisse zur Termination und Konfluenz von TES finden sich in [Dershowitz 82], [Dershowitz 85], [Drewes 89] und [Drewes/Lautemann 90] bzw. [Huet 77]. Aufgrund der Konfluenz solcher Systeme braucht ein Interpretierer für solche Systeme keine Konflikte zu lösen, da immer nur genau eine Lösung erzeugt wird, egal welcher Ableitungsweg gewählt wurde.

Für TES, die für die Verarbeitung natürlicher Sprache, also auch für die MÜ eingesetzt werden, können nicht beide Eigenschaften vorausgesetzt werden. Ein einfaches Beispiel macht deutlich, warum die Konfluenz nicht garantiert werden kann. In der syntaktischen Analyse ist aus einer syntaktisch mehrdeutigen Ausgangsstruktur mehr als eine äquivalente Zielstruktur ableitbar. Diese Tatsache kann zur Unvollständigkeit der Interpretation dieser TES führen, wenn die *Verzweigungspunkte* in einer Ableitung nicht durch eine revidierende Steuerung berücksichtigt werden. Aus diesem Grund werden die Verzweigungspunkte eines TES durch Überlagerung der linken Seiten aller TE-Regeln berechnet. Die Verzweigungspunkte werden durch die Steuerungsrelation $\vee_{alt}$ dargestellt.

Wenn TES für die Verarbeitung natürlicher Sprache eingesetzt werden, ist die Verwendung von *Defaultregeln* sinnvoll. Eine TE-Regel ist genau dann eine Defaultregel, wenn es mindestens eine TE-Regel mit einer spezifischeren linken Seite gibt, es sei denn, der Regelschreiber hat sie explizit als Verzweigungspunkte gekennzeichnet. Die Defaultregeln werden durch die Steuerungsrelation $<_{def}$ dargestellt. Existieren solche Defaultregeln und ist die spezifischere Regel auf einen Term anwendbar, so wird nur diese Regel angewendet und nicht die Defaultregel. Andernfalls wird die Defaultregel angewendet, falls sie anwendbar ist. Die Defaultregeln haben Einfluß auf die Korrektheit der Interpretation der TE-Regeln und müssen deshalb durch die Steuerung des Interpretierers gesondert behandelt werden.

Normalerweise wird durch die Anwendung einer TE-Regel die Voraussetzung für die Anwendung einer anderen geschaffen. In diesem Fall sind die beiden Regeln inhärent geordnet. Die Reihenfolgeabhängigkeiten werden durch die Steuerungsrelation $<_{app}$ dargestellt. Wenn die TE-Regeln genau in dieser Reihenfolge auf Anwendbarkeit geprüft werden, ist eine effiziente Anwendung gewährleistet. Allerdings muß berücksichtigt werden, daß diese *Anwendungsreihenfolge* in der Regel auch Zyklen enthält, welche die Effizienz je nach Anzahl und Größe der Zyklen mindern.

In [Weisweber 94a] und [Weisweber 92] wird ein effizienter, korrekter und vollständiger Algorithmus für die Interpretation von nicht-konfluenten TES beschrieben, der drei Steuerungsrelationen verwendet, die automatisch aus den TE-Regeln durch Überlagerung berechnet werden. Die Steuerungsrelationen explizieren genau die drei o.g. Beziehungen zwischen zwei TE-Regeln (Verzweigungspunkte, Defaultregeln und Anwendungsreihenfolge).

Die Terminierung von TES wird dort durch eine Termordnung gewährleistet, die auf disjunkte Vokabulare für Ausgangs- und Zielterme basiert. Der Interpretierer terminiert, weil durch die Anwendung einer TE-Regel das Ausgangsvokabular reduziert wird.

1.2.4 Das Anapherninterpretationsverfahren

Die Aufgabe des Verfahrens zur Anapherninterpretation⁶ besteht darin, für ein im Text auftretendes Personal- oder Possessivpronomen das intendierte Antezedens zu bestimmen. Als mögliche Antezedenten kommen alle Nominalphrasen, darunter auch andere Pronomina und koordinierte Nominalphrasen, in Frage, die im gleichen Satz und im Vortext auftreten. Die Länge des zu berücksichtigenden Vortexts ist als Parameter des Verfahrens gegeben. Nominalphrasen, die nach dem Pronomen im gleichen Satz auftreten, werden ebenfalls berücksichtigt. Dadurch werden neben intra- und intersententiellen anaphorischen Bezügen auch intrasententielle kataphorische Bezüge erfaßt.

In dem Verfahren zur Anapherninterpretation werden unterschiedliche Kriterien zur Bestimmung eines Antezedens integriert, indem sie alle im Sinne von Präferenzregeln aufgefaßt werden, d.h. als Kriterien, die einen Antezedenskandidaten als mehr oder weniger geeignet erscheinen lassen. Die aktuellen Kriterien sind **kongruenz**, **p.bindungen**, **naehe**, **subjekt**, **satzthema**, **rolle**, **freie_ergaenzung** und **diskurs**. Kriterien beziehen sich sowohl auf Eigenschaften, die dem Antezedenskandidaten allein zukommen (etwa Subjekt des Satzes zu sein), als auch auf Eigenschaften, die der Antezedenskandidat in Bezug auf das fragliche Pronomen hat (z.B. Nähe). Daneben kann es sich um strukturelle Eigenschaften wie die oben genannten handeln, aber auch um Eigenschaften der Referenzobjekte. Deshalb arbeitet das Verfahren auf einer dualen Repräsentation des Texts (siehe Abschnitt 1).

Die Kriterien können sich auf positive, aber auch auf negative Eigenschaften eines Antezedenskandidaten beziehen, und sie haben unterschiedlich starkes Gewicht. Dadurch können auch sogenannte harte Beschränkungen, etwa Bindung und Kongruenz, als Präferenzregeln integriert werden, nämlich als solche mit sehr starkem negativen Gewicht. Die Gewichte der Kriterien sind so gewählt, daß intuitiv konkurrierende Kriterien gleich starken Einfluß haben. Damit das Verfahren für experimentelle Zwecke geeignet ist, sind die Gewichte als Parameter formuliert.

Um den besten Antezedenskandidaten herauszufinden, bewertet das Verfahren alle für ein Pronomen in Frage kommenden Kandidaten anhand sämtlicher Kriterien. Aus Effizienzgründen kann es aber sinnvoll sein, sehr schlechte Kandidaten nicht weiter zu berücksichtigen. Dazu gibt es zwei Parameter, den Grenzwert und den Abstandswert. Der *Grenzwert* bildet eine absolute Grenze, unterhalb derer Kandidaten ausgeschlossen sind und im weiteren Verfahren nicht mehr bewertet werden. Der *Abstandswert* hat den gleichen Effekt wie der Grenzwert, bildet aber keine absolute, sondern eine relative Grenze, nämlich den zulässigen Abstand zum besten Kandidaten. Im Zusammenhang mit einer Möglichkeit zur Begrenzung wird auch die Anwendungsreihenfolge der Kriterien relevant. Kriterien mit stark negativem Gewicht sollten aus Effizienzgründen möglichst früh angewandt werden. Deshalb wird auch die Anwendungsreihenfolge durch Parameter festgelegt.

Das Verfahren bewertet alle für ein Pronomen in Frage kommenden Antezedenskandidaten anhand sämtlicher Kriterien. Der beste Kandidat wird dann als Antezedens gewählt. Pronomen und Antezedens verweisen auf dasselbe Referenzobjekt in der referentiellen Textrepräsentation. Deshalb wird vor allem dieser Verweis vom Antezedens für das Pronomen übernommen. Daneben werden einige semantische Eigenschaften übernommen und die Rollen des Pronomens in der Textrepräsentation realisiert.

Es kann vorkommen, daß es mehr als einen besten Kandidaten gibt oder auch keinen.

⁶Dieser Abschnitt entspricht sinngemäß dem Abschnitt 3.3 ab Seite 29 aus [KIT-FAST 93].

Im ersten Fall kann es sein, daß der Text schlecht gebaut, d.h. der anaphorische Bezug auch für einen menschlichen Leser ambig ist. Wahrscheinlicher ist es allerdings, daß die bisher entwickelten Kriterien nicht ausreichend sind. Das Verfahren müßte in diesem Fall interaktiv eine Entscheidung vom Benutzer anfordern. Zur Zeit wird allerdings einfach einer der Kandidaten gewählt. Der zweite Fall tritt auf, wenn es entweder gar keine An-tezedenskandidaten gibt, oder alle Kandidaten durch den Grenzwert ausscheiden. Dann wird das Pronomen als deiktisch interpretiert, d.h. es bildet ein eigenständiges Objekt in der referentiellen Textrepräsentation.

1.3 Strukturübergänge

Dieser Abschnitt geht näher auf die Strukturübergänge im MÜ-System ein, d.h. auf die morphologische Analyse und Synthese, auf die syntaktische, semantische und konzeptuelle Analyse, den Transfer und die Generierung.

Die morphologische Analyse und Synthese werden durch das im Abschnitt 1.2.1 beschriebene Verfahren durchgeführt und die syntaktischen Strukturen werden von dem im Abschnitt 1.2.2 konstruiert. Die restlichen vier Strukturübergänge werden durch einen Algorithmus auf der Basis von Termersetzung realisiert (siehe Abschnitt 1.2.3), der die in den Abschnitten 1.3.3 bis 1.3.6 beschriebenen Termersetzungssysteme interpretiert.

1.3.1 Die morphologische Analyse und Synthese

Der Parser und die morphologische Synthese des MÜ-Systems verwenden ein Stammformenlexikon. Die morphologische Analyse und Synthese basieren auf dem SUTRA-System, dessen Grundzüge im Abschnitt 1.2.1 beschrieben sind. Aus diesem Grund können dasselbe Stammformenlexikon, dasselbe morphologische Lexikon und dieselben morphologischen Daten (Flexionsparadigmen) sowohl für die Analyse als auch für die Synthese verwendet werden.

Die morphologische Analyse ermittelt für eine Wortform mithilfe eines Stammformenlexikons, eines morphologischen Lexikons und der Flexionsparadigmen einer Grammatik die entsprechende Kategorie mit den entsprechenden syntaktischen Merkmalspezifikationen. Dazu werden eine Präfix-, Infix- und Suffixanalyse der Wortformen durchgeführt. Die morphologische Synthese geht umgekehrt vor.

Die Morphologie des Deutschen geht von einer Klassifizierung der Nomen und Verben nach Deklinations- bzw. Flexionstyp aus und flektiert die Stammformen nach Kategorie, den vorgegebenen syntaktischen Daten (Person, Numerus, Genus, Kasus, Flektionsart, Tempus, Modus) und dem entsprechenden Typ.

Weiterhin wurde eine morphologische Komponente für das Englische auf der Basis der deutschen Morphologie entwickelt. Dabei wurden die Verben nach morphologischen Ähnlichkeiten bezüglich der Infinitiv-, Präteritum- und Partizipformen mit dem Ziel klassifiziert, diese verschiedenen Formen durch eine minimale Anzahl von Operationen zu flektieren. Die Operationen beziehen sich sowohl auf die Suffix- als auch auf Vokalveränderungen. Die Nomen wurden nach Plural- und Genitivformen klassifiziert. Präpositionen, Adjektive und Artikel werden als Vollformen in das englische Lexikon eingetragen.

1.3.2 Syntaktische Analyse

Im folgenden soll kurz auf die GPS-Grammatik des Deutschen, die im MÜ-System verwendet wird, eingegangen werden. Insgesamt umfaßt sie die nachstehend aufgeführten

Konstruktionen (vgl. [Preuß 87]):

- eine flache Satzstruktur mit dem Subjekt als subkategorisierter Kategorie
- Hauptsatzstellung mit einer topikalisierten Konstituente im Vorfeld
- eine hierarchische Hilfsverbstruktur
- Subjekt-kontrollierte Infinitivkonstruktionen
- *daß*-Sätze
- Relativsätze mit Relativpronomen im Nominativ
- einen Nominalkomplex mit den wichtigsten Phänomenen wie Determiner, Adjektiv, Nomen und Genitiv- und Präpositionalobjekten
- Abtrennbare Verbpräfixe
- Extraposition von Relativsätzen, *daß*-Sätzen und *um-zu*-Infinitiven
- Negationspartikel
- Einführung eines freien Adjunkts pro subkategorisiertem Vollverb

Die GPS-Grammatik des Deutschen, die im Projekt KIT-FAST verwendet wird, basiert auf [Uszkoreit 84]. Sie geht im wesentlichen von einer flachen Satzstruktur aus, in der das Verb und alle seine Argumente in einer ID-Regel vorkommen. Ein Fragesatz wird dann direkt durch eine solche flache Struktur repräsentiert. In einem Deklarativsatz wird eine Konstituente vorangestellt, d.h. mithilfe des Foot-Merkmals *slash*, des FFPs und einer *slash*-Einführungsmetaregel in die Position am Satzanfang gebracht. Die flache Satzstruktur kann ebenfalls als Schwesterkonstituente eines Hilfsverbs (Perfekt, Futur, Passiv oder Modal) vorkommen. Wenn mehrere Hilfsverben vorkommen, wird die flache Struktur zunächst mit dem Passivhilfsverb zu einer neuen satzwertigen Konstituente zusammengefaßt. Diese Konstituente wird dann mit dem Perfekthilfsverb ebenfalls zu einer satzwertigen Konstituente zusammengefaßt. Analoges gilt auch für das Modal- und das Futurhilfsverb. Wenn in einem Deklarativsatz Hilfsverben vorkommen, wird die zu topikalisierende Konstituente entsprechend weit transportiert.

Da die relativ einfache Beschreibung von Koordinations-Phänomenen als ein besonderer Vorteil des GPSG-Formalismus gilt, war es notwendig, diese Hypothese bei der deutschen Syntax zu überprüfen. In der ursprünglichen Version der GPSG aus [Gazdar et al. 85] werden alle Koordinationsregeln unter ein Schema zusammengefaßt. Wir mußten überprüfen, ob sich diese Regeln in der konstruktiven Version der GPSG unseres Projekts übernehmen ließen. Das Regelschema enthält eine iterative Regel mit Kleene-Star, auf die wir aus formalen Gründen verzichten, da die Umsetzung des Kleene-Stars in PS-Regeln zu einer Grammatik mit aufzählbar unendlich vielen Regeln führt, die nicht mehr kontextfrei ist. Desweiteren beschränken wir die Koordinationsregeln auf Rechts-Rekursivität, um Mehrfachanalysen zu vermeiden. Wir verwenden stark unterspezifizierte Regeln und kommen somit pro Konjunktion mit drei Regeln (davon eine rekursive Regel) aus.

Da in der ursprünglichen GPSG die koordinierten Kategorien nur von der *Head-Feature-Convention* (HFC) eingeschränkt werden, ergibt sich eine sehr tolerante Anwendung der Regeln, die z.B. auch die Koordination unterschiedlicher Hauptkategorien zulassen wie in *sie ist energisch und von großer beredsamkeit*. Die Unterspezifikation der Regeln

führt allerdings zu einem erheblichen Mehraufwand bei der automatischen Verarbeitung, da die Regeln auch in nicht koordinativen Strukturen zunächst anwendbar scheinen.

Die Änderung der Merkmalinstantiierungsprinzipien (siehe Abschnitt 1.1.2) war Voraussetzung dafür, die Koordination mit *und*, Objekt-kontrollierte Infinitive und Relativpronomen in allen Kasus in das Fragment aufzunehmen.

Die Nominalgruppensyntax umfaßt adjektivische und determinale Quantoren. Dabei wurde eine Determiner-Phrase-Analyse in Anlehnung an die Rektions- und Bindungstheorie von Chomsky für den Nominalkomplex eingeführt.

Eine ausführliche Beschreibung der Grammatiken geben [Preuß 87], [Preuß 89] und zum Teil auch [Busemann 92].

1.3.3 Die semantische Analyse

Die semantische Analyse bildet GPSG-Strukturen des Deutschen in FAS-Ausdrücke des Deutschen ab. Die Strategie der Abarbeitung der Ausgangsstruktur wird im wesentlichen durch folgende Probleme bestimmt:

- Topikalisierte oder extraponierte Konstituenten sind Tochterknoten des Wurzelknotens der GPSG-Struktur und treten in der FAS in der zugehörigen Funktor-Argument-Ebene auf (impliziert top-down Strategie).
- Die Abbildung der Oberflächenkasus in die Tiefenkasus (semantische Rollen) kann erst nach dem “Rücktransport” der topikalisierten bzw. extraponierten Konstituenten erfolgen, da erst dann alle zugehörigen Funktoren bzw. Argumente vorhanden sind.
- Possessivpronomen sind in der FAS Argumente des nominalen Prädikates (Hauptkategorie *n-pred*). In der syntaktischen Struktur werden sie als Determinatoren aufgefaßt und müssen deshalb innerhalb der Nominalphrase an beliebig vielen Modifikatoren und Relativsätzen vorbei bis zum Nomen “transportiert” werden (impliziert top-down Strategie).
- Die Negation wird immer als Satznegation aufgefaßt und in der FAS in der obersten *clause*-Konstruktion repräsentiert (impliziert bottom-up Strategie).
- Informationen über Tempus und Modus stehen in der GPSG-Struktur am Hilfs- oder Vollverb. In der FAS werden sie an einem höheren Knoten repräsentiert (impliziert bottom-up Strategie).

Aus den Erfordernissen der semantischen Analyse ergibt sich somit die Tatsache, daß sie nur in zwei Phasen durchgeführt werden kann. In der ersten Phase wird die GPSG-Struktur bottom-up abgearbeitet und in der anschließenden Phase top-down. Außerdem benötigt man für die Abbildung des Oberflächenkasus in den Tiefenkasus in der bottom-up Phase sowohl syntaktische als auch semantische Informationen. Beide Gründe legen nahe, eine Zwischenebene einzuführen, welche die Repräsentation hybrider (syntaktischer und semantischer) Informationen erlaubt.

In der ersten Phase, in der bottom-up vorgegangen wird, werden die ausgangssprachlichen Kategorien durch hybride Kategorien und anschließend während der top-down Phase die hybriden Kategorien durch zielsprachliche Kategorien ersetzt. Die hybride Ebene tritt nach außen hin nicht in Erscheinung. Die Termordnung fordert dann, daß in einer TE-Regel eine ausgangssprachliche Kategorie gelöscht oder durch mindestens eine hybride

oder zielsprachliche Kategorie ersetzt wird. Eine andere Möglichkeit ist, eine hybride Kategorie zu löschen oder durch mindestens eine zielsprachliche Kategorie zu ersetzen. Letzteres muß gelten, wenn auf der linken und rechten Seite einer TE-Regel die gleiche Anzahl von Kategorien der Ausgangssprache vorkommt.

In der ersten Phase werden das Negations-Partikel und die Information über Perfektiv, Passiv, Futur, Präsens/Imperfekt und Indikativ/Konjunktiv nach oben “transportiert”, um sie am obersten *v-pred-*, *clause-* bzw. *clausefin*-Knoten zu repräsentieren.

In der zweiten Phase wird dann, falls vorhanden, eine topikalisierte und/oder extrapolierte Konstituente, die am Wurzelknoten der GPSG-Struktur steht, an ihre kanonische Position innerhalb der FAS entlang des Wertes für das Merkmal *slash* in der Struktur zurückgeführt und anschließend die Oberflächenkasus in die entsprechenden semantischen Rollen übersetzt. Diese Abbildung erfolgt mithilfe einer Zuordnungstabelle, die Daten über syntaktische und semantische Subkategorisierung und Oberflächen- und Tiefenkasus enthält. Auf sie kann mithilfe eines Prädikates im Bedingungsteil einer TE-Regel zugegriffen werden. Die Information über das Genus Verbi, die ebenfalls für die Abbildung benötigt wird, ist implizit in der syntaktischen Subkategorisierung enthalten. Außerdem werden Possessivpronomen in die Argumentposition des entsprechenden nominalen Prädikates transportiert.

Da die Information über Indikativ oder Konjunktiv nicht in der GPSG-Struktur repräsentiert wird, werden in der semantischen Analyse alle Sätze in den Indikativ abgebildet. Dieses Problem läßt sich aber durch Einführung eines entsprechenden Merkmals in der GPSG beheben.

1.3.4 Die konzeptuelle Analyse

Die Ausgangsstrukturen der konzeptuellen Analyse sind FAS-Ausdrücke des Deutschen, die in ATL-Strukturen des Deutschen abgebildet werden. Dies wird am Beispiel der Ableitung der ATL-Struktur in Abbildung 6, die hier als Abbildung 8 wiederholt ist, aus der FAS in Abbildung 7 demonstriert.

Die konzeptuelle Analyse erfolgt im wesentlichen bottom-up und in einer Phase. Aufgrund der großen strukturellen Ähnlichkeit von FAS-Ausdrücken und ATL-Strukturen besteht die konzeptuelle Analyse überwiegend aus der Reduktion der FAS-Ausdrücke. Dabei werden aus den FAS-Ausdrücken die Referenzen auf ABox-Objekte, die Diskursobjekte repräsentieren, und die semantischen Rollen in den ATL-Strukturen übernommen. Die Konzepte der Diskursobjekte werden in der TBox repräsentiert. Die TBox-Konzepte werden durch Zugriff auf ein FAS-ATL-Lexikon aus den FAS-Terminalen und -Präterminalen ermittelt. Die Referenzen, semantischen Rollen und TBox-Konzepte werden durch die Merkmale *obj-ref*, *inv/role* bzw. durch die Terminale *<concept>* der Syntax für ATL-Strukturen in Abbildung 3 repräsentiert.

Die FAS-Ausdrücke werden bottom-up abgearbeitet, d.h. zunächst werden die terminalen und preterminalen Symbole der FAS ersetzt. Abbildung 9 gibt die TE-Regeln an, die in ihrem Bedingungsteil auf das Abbildungslexikon für den Übergang von der semantischen zur konzeptuellen Repräsentation zugreifen. Anschließend folgen die strukturellen TE-Regeln, von denen Abbildungen 10 und 11 zwei Beispiele sind. Sie bilden die Argumente mit der Hauptkategorie *term* bzw. den obersten Knoten der FAS, der immer die Hauptkategorie *fas* hat, ab.

Die bottom-up Verarbeitung der FAS ist implizit in den TE-Regeln kodiert. Die Berechnung der Reihenfolgerelation (siehe Abschnitt 1.2.3) ergibt, daß die lexikalischen

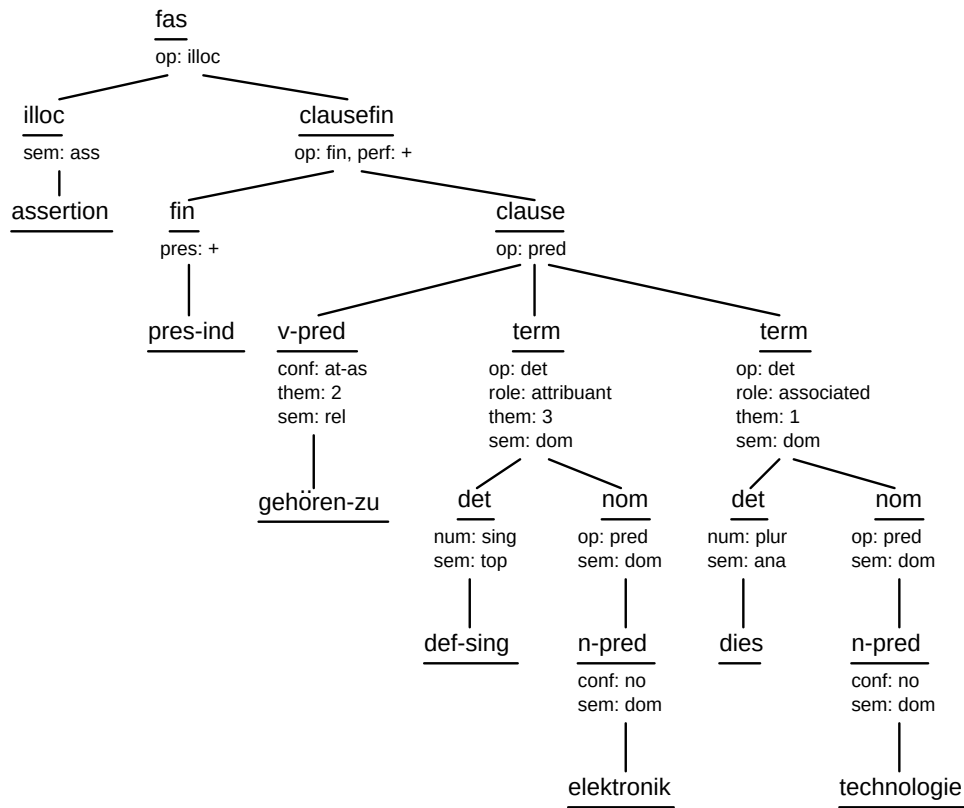


Abbildung 7: FAS-Ausdruck für *zu diesen technologien gehört die Elektronik*

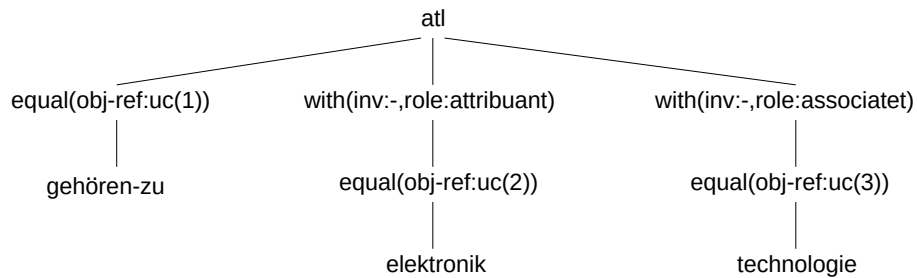


Abbildung 8: ATL-Struktur für *zu diesen technologien gehört die Elektronik*

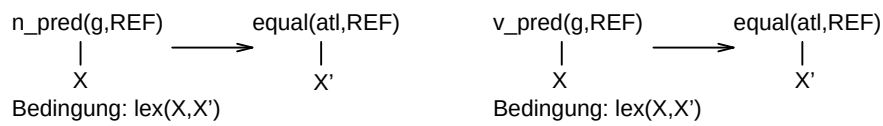


Abbildung 9: Lexikalische TE-Regeln in der konzeptuellen Analyse

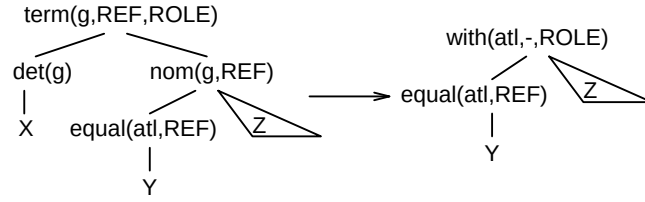


Abbildung 10: TE-Regel für die Abbildung eines Arguments

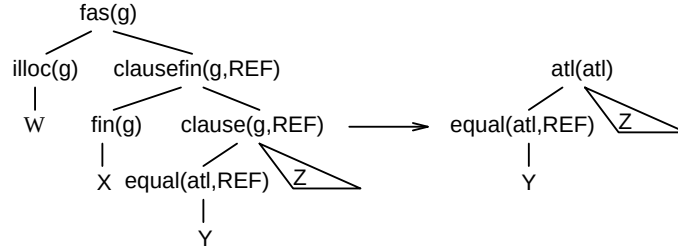


Abbildung 11: TE-Regel für die Abbildung des obersten FAS-Knotens

TE-Regeln in Abbildung 9 vor den beiden anderen TE-Regeln angewendet werden, die in der Reihenfolge unabhängig sind.

Nach der konzeptuellen Analyse bildet eine Post-Routine die ATL-Struktur eins-zu-eins in ATL-Ausdrücke gemäß der Syntax des BACK-Systems ab (siehe Abschnitt 1.1.4). Diese Routine extrahiert Pronomen, die zunächst nicht in die ABox eingetragen werden dürfen, da die Referenz für die jeweiligen Antezedenten durch die Auswertungskomponente für anaphorische Bezüge ermittelt werden muß, bevor die Informationen darüber in die ABox eingetragen werden dürfen. Andernfalls würden die Pronomina vom BACK-System als neue ABox-Objekte eingetragen. Dies wird nur bei Pronomina gemacht, für die kein Antezedent gefunden wird. Sie werden quasi als deiktische Anapher aufgefaßt.

Die Informationen über Relativpronomen und implizite Subjekte, die sich auf ein Pronomen beziehen, werden wie die Information über Pronomina erst nach Abschluß der Auswertung der anaphorischen Bezüge durch die Anapherninterpretationskomponente in die ABox eingetragen. Die Information wird dabei unter dem ABox-Objekt eingetragen, welches das Antezedens repräsentiert (siehe Abschnitt 1.1.4).

Sämtliche Informationen, die sich nicht auf Pronomina beziehen, werden vor Beginn der Auswertungskomponente in die ABox eingetragen, damit für die Bewertung der semantischen Konsistenz möglicher Antezedenten die maximal verfügbare Information in der ABox vorhanden ist.

Individuale Konzepte wie zum Beispiel Eigennamen werden vor dem Eintrag in die ABox durch Anfragen an die TBox und ABox mithilfe des BACK-Systems überprüft. Wenn ein individuelles Konzept bereits eine Referenznummer in der ABox hat, werden die entsprechenden Informationen unter dieser Referenz eingetragen. Andernfalls kommen sie im Diskurs das erste Mal vor und bekommen eine neue Referenznummer.

Nach der Auflösung der anaphorischen Bezüge und dem Eintrag der Informationen, die sich auf anaphorische Ausdrücke bezieht, in die ABox sind alle Variablen für die Werte des Merkmals *ref* im entsprechenden FAS-Ausdruck durch Variablenbindung instantiiert und stehen für den Transfer zur Verfügung.

1.3.5 Der Transfer

Die Ausgangsstrukturen des Transfers sind FAS-Ausdrücke des Deutschen und die Zielstrukturen FAS-Ausdrücke des Englischen. Die Ausgangsstruktur wird im wesentlichen bottom-up und von rechts nach links abgearbeitet, d.h. die lexikalischen Transferregeln werden vor den strukturellen angewendet und die Argumente werden vor den Funktoren ersetzt. Von den 250 Transferregeln sind der überwiegende Teil lexikalisch (ca. 80%). Aufgrund der Tatsache, daß die FAS auf die speziellen Bedürfnisse des Transfers zugeschnitten wurde und die FAS des Deutschen und Englischen bis auf die Terminalsymbole sehr ähnlich sind, werden relativ wenig strukturelle Transferregeln benötigt.

Strukturelle Transferregeln sind für die eins-zu-eins Abbildungen und für strukturelle Änderungen notwendig. Da die ersteren relativ uninteressant sind, gehe ich im folgenden nur auf die Transferregeln ein, die strukturelle Änderungen vornehmen:

- Wenn in einem deutschen Satz ein Argument eines Verbs in der thematischen Struktur vor dem Subjekt vorkommt, so wird, falls möglich, das Verb im Englischen passiviert oder eine lexikalische Konverse als Übersetzung gewählt, um die thematische Satzstruktur so weit wie möglich beizubehalten.
- Wird ein Nomen in ein Singulartantum (engl. *mass noun*) übersetzt, wird ein evtl. vorkommender Artikel in einen indefiniten Artikel übersetzt, der auf der Oberfläche nicht realisiert wird. Falls ein deutsches Nomen im Plural steht wird alternativ dazu auch die Übersetzung in *pieces of* + Nomen zugelassen.
- Das Wort *viel* wird im Deutschen als Adjektiv und im Englischen als Artikel behandelt. In Verbindung mit einem aufzählbaren Nomen (engl. *count noun*) wird es in *many* und mit Singulariatantum in *much* übersetzt.
- Die Übersetzung von Funktoren (Verben, Nomen und Adjektive) wird in der Regel von den semantischen Rollen und Merkmalen der Argumente abhängig gemacht. Dabei findet zum Teil ein Wechsel der Werte dieser semantischen Merkmale statt. Dieses Phänomen wurde aber nicht näher systematisiert.
- Einige feste Syntagmen wurden sowohl auf der ausgangs- als auch auf der zielsprachlichen Seite berücksichtigt, wie z.B. *grundlagenforschung* in *basic research*, *außerdem* in *in addition*, *abstützen auf* in *to be based on*, *in ihrer größenordnung zu gering sein* in *to be too small*, *gefährden* in *to be at stake* und *abhängen von* in *to be dependant on*. Wenn ein festes Syntagma (z.B. ein Funktionsverbgefüge) in der Ausgangssprache vorkommt, so kann es adäquat mit Defaultregeln behandelt werden. Die Regel, deren linke Seite das feste Syntagma beschreibt ist spezifischer als die der Transferregel, welche die Übersetzung des Funktionsverbs allein beschreibt.
- Idiome in der Ausgangssprache können analog zu festen Syntagmen adäquat mit Defaultregeln übersetzt werden.
- Falls Sätze mit der gleichen subordinierenden Konjunktion koordiniert werden, wird in der Zielsprache nur eine Konjunktion eingeführt und die beiden Sätze koordiniert.
- Das Demonstrativpronomen *diese* wird in das englische anaphorische Adjektiv *such* übersetzt, wenn das determinierte Nomen im Englischen zu einer Klasse gehört, bei der der Artikel wegfällt, z.B. *diese technologien* in *such technologies*.

- Kommt im Deutschen das Adjektiv *solch* in Verbindung mit einem indefiniten Artikel vor, z.B. *eine solche projektbeteiligung*, so wird der indefinite Artikel im Englischen auf der Oberfläche nicht realisiert (*such coparticipation*).
- Im Englischen wird ein Merkmal für die progressive Form eingeführt. Dabei wird das Präsens im Deutschen entweder in das Präsens oder das Progressiv übersetzt, so daß in diesem Fall zwei Übersetzungen resultieren.

1.3.6 Die Generierung

Die Ausgangsstruktur der Generierung ist ein englischer FAS-Ausdruck und die Zielstruktur eine entsprechende syntaktische Struktur. Die Probleme dabei sind im großen und ganzen dieselben wie in der semantischen Analyse, nur umgekehrt.

- Ein Argument aus der Funktor-Argument-Ebene muß als Subjekt der syntaktischen Struktur realisiert werden.
- Falls erforderlich, muß ein Argument des Verbs topikalisiert werden.
- Die semantischen Rollen müssen in die entsprechenden Oberflächenkasus abgebildet werden.
- Die Possessivpronomina, die in der FAS als Argument des Nomens auftreten, werden zum Determinator des Nomens.
- Die Informationen über Perfekt, Progressiv und Futur sind in der FAS als Merkmalspezifikationen repräsentiert und dafür müssen entsprechende Hilfsverben erzeugt werden.

Die Generierung arbeitet im wesentlichen top-down. Da aber für die Abbildung der semantischen Rollen in die Oberflächenkasus an den Funktoren hybride Informationen (semantische Konfiguration und syntaktische Subkategorisierung) notwendig sind, werden zunächst die Verben, Nomen und Adjektive in entsprechende hybride Funktoren abgebildet. Erst danach wird die Struktur top-down abgearbeitet, indem zunächst der oberste Satz-knoten eingeführt wird. Anschließend wird überprüft, ob ein Argument zu topikalisieren bzw. extraponieren ist. Die zu topikalisierende bzw. extraponierende Konstituente darf nicht den Kasus Nominativ bekommen. Für die Einführung des Subjekts ist eine andere TE-Regel zuständig. Sie führt die Nominalphrase im Nominativ oder des expletive *it* bzw. *there* als Subjekt ein. Die Kasus werden durch einen Zugriff auf die Kasus-Rollen-Tabelle über den Bedingungsteil der TE-Regeln ermittelt. Falls eine Konstituente weder zu topikalisieren noch zu extraponieren ist, wird nur das Subjekt des Satzes eingeführt. Dieses läßt sich dadurch erkennen, daß das Argument, das als Oberflächenkasus den Nominativ bekommt, in der thematischen Struktur des Satzes an erster Stelle vorkommt. Danach werden die erforderlichen Hilfsverben eingeführt. Es gibt 8 TE-Regeln, welche die Hilfsverben für Futur, Perfekt und Progressiv generieren. Es sind insgesamt 8 TE-Regeln notwendig, um die Termordnung einhalten zu können, weil nur ein Knoten des FAS-Ausdrucks ersetzt werden kann. Da die Hilfsverben durch drei zweiwertige Merkmale kodiert sind, gibt es 8 Kombinationen. Anschließend werden Modalverben realisiert, falls vorhanden, und das Passivhilfsverb an unterster Stelle nach den anderen Hilfsverben eingeführt. Nachdem alle Hilfsverben generiert wurden, werden die semantischen Rollen der Argumente des Verbs in die jeweiligen Oberflächenkasus abgebildet. Die entsprechende

Regel greift über den Bedingungsteil auf die Kasus-Rollen-Tabelle zu. Solche Regeln sind ebenfalls für Nomen und Adjektive definiert.

Possessivpronomina sind in der FAS als Argumente des Nomens realisiert. Dies erlaubt eine einheitliche Behandlung von Personal- und Possessivpronomina in der Anaphernresolution. Deshalb wird ein Possessivpronomen durch eine TE-Regel von der Argumentposition des Nomens in die Determinatorposition transportiert. Dabei erfolgt der Transport an beliebig vielen Modifikatoren vorbei innerhalb des nominalen Komplexes. Falls kein Possessivpronomen vorhanden ist, wird ein Artikel gemäß der Spezifikationen in der FAS generiert. Die entsprechende TE-Regel und diejenige für die Abbildung des Possessivpronomens bilden einen Verzweigungspunkt. Dies entspricht der Tatsache, daß ein Possessivpronomen sowohl als Determinator als auch als Präpositionalattribut des Nomens realisiert werden kann (*his* vs. *of him*). Gleichzeitig ist die linke Seite der Regel für die Abbildung des Determinators allgemeiner als die für die Abbildung des Possessivpronomens, so daß beide Regeln außerdem in Defaultrelation stehen. Hier muß der Regelschreiber entscheiden, ob ein Verzweigungspunkt erzeugt werden soll oder nicht. Für das im MÜ-System eingesetzte TE-System wurde der Verzweigungspunkt erzeugt, so daß immer zwei Lösungen erzeugt werden, wenn ein Possessivpronomen in einem FAS-Ausdruck vorkommt.

2 Installationshandbuch

In diesem Abschnitt wird beschrieben, wie das experimentelle MÜ-System des Projekts KIT-FAST erhältlich ist. Es steht über einen FTP-Server und auf Disketten kostenlos zur Verfügung. Anschließend werden die Installation und die Struktur des Verzeichnisses erläutert, welches die Prolog-Programme für das MÜ-System, linguistische Daten für die Übersetzungsrichtung Deutsch → Englisch, Werkzeuge zum Erstellen der linguistischen Daten und Dokumentationen enthält. Im dritten Abschnitt werden die Prolog-Programme für die Unifikation und für den Interpretierer nicht-konfluenter Termersetzungssysteme (TES) angegeben. Diese Programme werden in leicht veränderter Form im MÜ-System verwendet. Abschließend wird die interne Darstellung der linguistischen Daten für jede Komponente des MÜ-Systems vorgestellt.

2.1 Installation des MÜ-Systems

Das MÜ-System wurde in Quintus-Prolog 3.1.4 unter SunOS 4.1.3 entwickelt. Es benötigt auf der Festplatte ca. 9.5 MB (in komprimierter Form ca. 3 MB) freien Speicherplatz.

Das MÜ-System läuft auch in SWI-Prolog. SWI-Prolog ist eine Public Domain Implementation von Prolog, die von Jan Wielemaker an der Universität Amsterdam (E-Mail-Kontakt: jan@swi.psy.uva.nl) entwickelt wurde. SWI-Prolog ist erhältlich über den FTP-Server [swi.psy.uva.nl](ftp.swi.psy.uva.nl).

Das MÜ-System ist kostenlos über den FTP-Server <ftp.cs.tu-berlin.de> oder auf drei Disketten erhältlich.

Dateien auf dem FTP-Server <ftp.cs.tu-berlin.de>

Das Verzeichnis `pub/doc/reports/tu-berlin.de/kit/FAST-v94` auf dem FTP-Server enthält folgende Dateien:

- `FILES.ENG` (Struktur des Verzeichnisses mit dem MÜ-System in englischer Sprache)
- `FILES.GER` (Struktur des Verzeichnisses mit dem MÜ-System in deutscher Sprache)
- `README.ENG` (Inhalt dieses Abschnitts in englischer Sprache)
- `README.GER` (Inhalt dieses Abschnitts in deutscher Sprache)
- `v94.tar.Z.aa` (Verzeichnis mit dem MÜ-System, 1. Teil)
- `v94.tar.Z.ab` (Verzeichnis mit dem MÜ-System, 2. Teil)
- `v94.tar.Z.ac` (Verzeichnis mit dem MÜ-System, 3. Teil)

Dateien auf den drei Disketten

Die Disketten sind zu beziehen beim Autor, TU Berlin, FLP, Sekr. FR 6-10, Franklinstr. 28/29, D-10587 Berlin. Sie enthalten folgende Dateien:

- Diskette 1:
 - `FILES.ENG` (Struktur des Verzeichnisses mit dem MÜ-System in Englisch)

- FILES.GER (Struktur des Verzeichnisses mit dem MÜ-System in Deutsch)
- README.ENG (Inhalt dieses Abschnitts in englischer Sprache)
- README.GER (Inhalt dieses Abschnitts in deutscher Sprache)
- v94.tar.Z.aa (Verzeichnis mit dem MÜ-System, 1. Teil)
- Diskette 2:
 - v94.tar.Z.ab (Verzeichnis mit dem MÜ-System, 2. Teil)
- Diskette 3:
 - v94.tar.Z.ac (Verzeichnis mit dem MÜ-System, 3. Teil)

Um das MÜ-System auf der Festplatte zu installieren, müssen die folgenden Befehle nacheinander ausgeführt werden (nach den Befehlen ist sowohl im Sun-Betriebssystem (SunOS) als auch im Prolog-Interpreter immer die Return-Taste zu drücken).

Kopieren vom FTP-Server

1. Lokales Verzeichnis einrichten, z.B.: `mkdir ~/FAST-v94`
2. Verbindung zum FTP-Server: `ftp ftp.cs.tu-berlin.de`
3. Name: `anonymous`
4. Passwort: `<eigene E-Mail-Adresse>`
5. In das Verzeichnis mit dem MÜ-System auf dem FTP-Server wechseln:
`cd pub/doc/reports/tu-berlin.de/kit/FAST-v94`
6. Lokales Verzeichnis festlegen: `lcd ~/FAST-v94`
7. Dateien kopieren: `mget *`
8. Frage: `mget FILES.ENG?` `y`
9. Frage: `mget FILES.GER?` `y`
10. Frage: `mget README.ENG?` `y`
11. Frage: `mget README.GER?` `y`
12. Frage: `mget v94.tar.Z.aa?` `y`
13. Frage: `mget v94.tar.Z.ab?` `y`
14. Frage: `mget v94.tar.Z.ac?` `y`
15. FTP-Programm verlassen: `quit`
16. In das lokale Verzeichnis wechseln: `cd ~/FAST-v94`
17. Weiter bei **Wiederherstellen des Verzeichnisses mit dem MÜ-System**

Kopieren von den Disketten

1. Verzeichnis einrichten, z.B.: `mkdir ~/FAST-v94`
2. In das Verzeichnis wechseln: `cd ~/FAST-v94`
3. Diskette 1 in das Laufwerk `/dev/rfd0c` stecken
4. Diskette 1 kopieren: `tar -xvf /dev/rfd0c`
5. Diskette 1 auswerfen: `eject`
6. Diskette 2 in das Laufwerk `/dev/rfd0c` stecken
7. Diskette 2 kopieren: `tar -xvf /dev/rfd0c`
8. Diskette 2 auswerfen: `eject`
9. Diskette 3 in das Laufwerk `/dev/rfd0c` stecken
10. Diskette 3 kopieren: `tar -xvf /dev/rfd0c`
11. Diskette 3 auswerfen: `eject`
12. Weiter bei **Wiederherstellen des Verzeichnisses mit dem MÜ-System**

Wiederherstellen des Verzeichnisses mit dem MÜ-System

1. Dateien zusammenfügen, dekomprimieren und das Verzeichnis wiederherstellen:
`cat v94.tar.Z.* | uncompress | tar -xvf -`
2. In das MÜ-System-Verzeichnis wechseln: `cd ~/FAST-v94/prog/mt`
3. Prolog aufrufen, etwa mit: `prolog7`

Durch die Eingabe des letzten Befehls wird der Prolog-Interpreter aufgerufen. Nach dem Start fordert der Interpreter durch den Prompt '?' zur Eingabe auf und das MÜ-System kann durch die Eingabe `[loadmt].` `Return` geladen und gestartet werden. Die Laderoutine in der Datei `~/FAST-v94/prog/mt/loadmt.pl` stellt fest, welcher Interpreter vorliegt und lädt die entsprechende Datei `~/FAST-v94/prog/mt/loadmtQUI.pl` oder `~/FAST-v94/prog/mt/loadmtSWI.pl`. In beiden Dateien ist festgelegt, in welcher Reihenfolge die Quell-Dateien des MÜ-Systems geladen werden. Diese Dateien können ersatzweise auch durch `[loadmtQUI], main.` `Return` bzw. `[loadmtSWI], main.` `Return` im jeweiligen Interpreter geladen werden (durch das Prädikat `main/0` wird das MÜ-System gestartet). Der Quell-Kode wird in der folgenden Reihenfolge geladen:

1. Das BACK-System (ein Wissensrepräsentationssystem zur Repräsentation des Text- und Hintergrundwissens als assertionales bzw. terminologisches Wissen, siehe [Hoppe et al. 93] und [Quantz/Kindermann 90]).

⁷Der lokale Befehl zum Starten des Quintus- oder SWI-Prolog-Interpreters ist den Systemhandbüchern zu entnehmen oder beim Systemadministrator zu erfragen. Voraussetzung ist die Installation eines der beiden Interpreter auf dem lokalen Rechner. Der Interpreter kann dann direkt aus dem Prolog-Verzeichnis oder durch das Setzen eines entsprechenden Pfades aufgerufen werden.

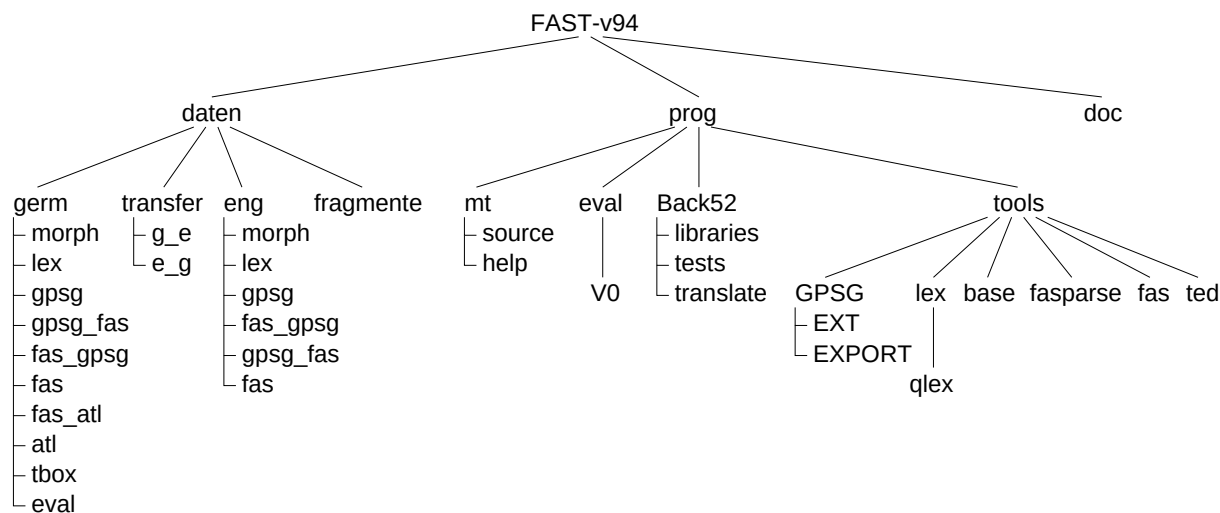


Abbildung 12: Struktur des Verzeichnisses `~/FAST-v94`

2. Prolog-Systemprädikate.
3. Das MÜ-System.
4. Die Anapherninterpretationskomponente (siehe [Dunker/Umbach 93]).

Nach dem Start des MÜ-Systems wird abgefragt, welche Übersetzungsrichtung geladen werden soll. Für das gelieferte MÜ-System ist ausschließlich die Richtung Deutsch → Englisch (Eingabe `g-e` `Return`) sinnvoll. Anschließend wird gefragt, ob die Dateien mit den linguistischen Defaultdaten, die in der Datei `~/FAST-v94/prog/mt/mtmenu.pl` festgelegt sind, geladen werden sollen oder nicht. Hier ist es sinnvoll, diese Frage mit ja (Eingabe `j` `Return`) zu beantworten, da sonst alle Dateien einzeln eingegeben werden müssen. Danach werden die Dateien mit den linguistischen Daten geladen (Details siehe Abschnitt 3.5.2 auf Seite 99).

Nach dem Laden der linguistischen Daten wird der Prompt `MT(AS --> ZS)>` des MÜ-Systems ausgegeben, wobei *AS* die Ausgangssprache und *ZS* die Zielsprache spezifizieren. Der MÜ-System-Interpreter erlaubt eine komfortablere Eingabe als der Prolog-Interpreter. Er ist im Abschnitt 3.2 beschrieben.

2.2 Die Verzeichnisstruktur

Die Daten für das MÜ-System umfassen die Prolog-Programme für das MÜ-System, linguistische Daten für die Übersetzungsrichtung Deutsch → Englisch, Werkzeuge zum Erstellen der linguistischen Daten und Dokumentationen. Abbildung 12 zeigt die Struktur des Verzeichnisses mit sämtlichen Daten. Das Verzeichnis ergibt sich, nachdem die drei Bandarchivierungsdateien `v94.tar.Z.aa`, `v94.tar.Z.ab` und `v94.tar.Z.ac` zusammengefügt, dekomprimiert und das Verzeichnis wiederhergestellt wurde (siehe Punkt 1 auf Seite 31 *Wiederherstellen des Verzeichnisses mit dem MÜ-System*).

2.2.1 Das MÜ-System

- Das Verzeichnis `mt` enthält das MÜ-System

- Das Verzeichnis **source**: Prolog-Programme für das MÜ-System
 - Das Verzeichnis **help**: Dateien für die Online-Dokumentation der MÜ-System-Befehle (vgl. Abschnitt 3.2)
 - Die Datei **loadmt.pl**: Datei zum Laden der Prolog-Programme für das MÜ-System (lädt **loadmtQUI.pl** oder **loadmtSWI.pl** und startet das MÜ-System)
 - Die Datei **loadmtQUI.pl**: Datei zum Laden der Prolog-Programme für das MÜ-System in Quintus-Prolog
 - Die Datei **loadmtSWI.pl**: Datei zum Laden der Prolog-Programme für das MÜ-System in SWI-Prolog
 - Die Datei **mtmenu.pl**: Spezifikation der Defaultdateien mit den linguistischen Daten für das MÜ-System (siehe Abschnitt 3.5.1 und Anhang B)
 - Die Datei **synonyms.pl**: Datei zur Definition von Synonymen (Abkürzungen) für Prädikatnamen (siehe Abschnitt 3.2.2 und Anhang C)
 - Die Datei **noback.pl**: Datei zum Ausschluß von (MÜ-)System-Prädikaten vom Backtracking (siehe Abschnitt 3.2.3 und Anhang D)
- Das Verzeichnis **eval**: Modul für das Anapherninterpretationsverfahren (siehe [Dunker/Umbach 93])
 - Das Verzeichnis **Back52**: Wissensrepräsentationssystem BACK (siehe [Hoppe et al. 93] und [Quantz/Kindermann 90])

2.2.2 Die Daten für das MÜ-System

- Das Verzeichnis **germ** enthält linguistische Daten für das Deutsche
 - Das Verzeichnis **morph**: Morphologie
 - Das Verzeichnis **lex**: GPSG- und FAS-Lexika
 - Das Verzeichnis **gpsg**: Daten für die syntaktische Analyse (GPSG)
 - Das Verzeichnis **gpsg_fas**: Daten für die semantische Analyse
 - Das Verzeichnis **fas_gpsg**: Daten für die Generierung
 - Das Verzeichnis **fas**: Daten für die semantische Repräsentation (FAS)
 - Das Verzeichnis **fas_atl**: Daten für die konzeptuelle Analyse
 - Das Verzeichnis **atl**: Daten für die konzeptuelle Repräsentation (ATL)
 - Das Verzeichnis **tbox**: Hintergrundwissen (TBox) für die Textrepräsentation
 - Das Verzeichnis **eval**: Daten für die Anapherninterpretation
- Das Verzeichnis **transfer** enthält linguistische Daten für den Transfer
 - Das Verzeichnis **g_e**: Daten für den Transfer Deutsch → Englisch
 - Das Verzeichnis **e_g**: Daten für den Transfer Englisch → Deutsch
- Das Verzeichnis **eng** enthält linguistische Daten für das Englische
 - Das Verzeichnis **morph**: Morphologie

- Das Verzeichnis **lex**: GPSG- und FAS-Lexika
- Das Verzeichnis **gpsg**: Daten für die syntaktische Analyse (GPSG)
- Das Verzeichnis **fas_gpsg**: Daten für die Generierung
- Das Verzeichnis **gpsg_fas**: Daten für die semantische Analyse
- Das Verzeichnis **fas**: Daten für die semantische Repräsentation (FAS)
- Das Verzeichnis **fragmente** enthält das bearbeitete Textfragment
 - Die Datei **frag92**: Beispielsätze in Deutsch und Englisch
 - Die Datei **gpsg92**: GPSG-Strukturen des Deutschen für die Beispielsätze
 - Die Datei **fas92**: FAS-Ausdrücke des Deutschen für die Beispielsätze
 - Die Datei **eval_fas92**: ausgewertete FAS-Ausdrücke des Deutschen
 - Die Datei **gen92**: FAS-Ausdrücke des Englischen für die Beispielsätze

2.2.3 Werkzeuge zum Erstellen der linguistischen Daten

Das Verzeichnis **tools** enthält Werkzeuge zur Entwicklung von linguistischen Daten für das MÜ-System. Es existieren Werkzeuge für folgende Komponenten:

- Das Verzeichnis **GPSG**: Compiler für *GPS-Grammatiken*, der die einzelnen Komponenten einer GPSG (Kategoriendefinitionen, FCRs, Aliases, ID-Regeln inkl. Metaregeln und LP-Aussagen) von einem lesbaren externen Format (Extension **.ext**) in das entsprechende interne, für das MÜ-System lesbare Format (Extension **.int**) übersetzt.
- Das Verzeichnis **lex**: Compiler für *Übergangslexika* $GPSG \rightarrow FAS$, die für die syntaktische und semantische Analyse benötigt werden, und *Übergangslexika* $FAS \rightarrow TBox$, die für die konzeptuelle Analyse benötigt werden. Der Compiler übersetzt die Lexika von einem lesbaren externen Format (Extension **.ext**) in das entsprechende interne, für das MÜ-System lesbare Format (Extension **.int**).
- Das Verzeichnis **fas**: Compiler für *FAS-Grammatiken*, mit *FAS-Kategoriendefinitionen* und *kontextfreien FAS-Regeln*. Der Compiler übersetzt eine FAS-Grammatik von einem lesbaren externen Format (Extension **.ext**) in das entsprechende interne, für das MÜ-System lesbare Format (Extension **.int**).
- Das Verzeichnis **fasparse**: Parser für terminale Eingabeketten, der mithilfe von FAS-Kategoriendefinitionen, einer FAS-Grammatik und eines Übergangslexikons $FAS \rightarrow TBox$ (s.o.) die entsprechenden *FAS-Ableitungsbäume* (*FAS-Terme*) erzeugt. Diese Strukturen können vom MÜ-System verarbeitet werden (siehe beispielsweise die FAS-Terme in den Dateien **fas92**, **eval_fas92** oder **gen92** im Verzeichnis **fragmente**).
- Das Verzeichnis **ted**: Editor für Termersetzungssysteme (TED), die für alle Übergänge außer der morphologischen und syntaktischen Analyse sowie der morphologischen Synthese verwendet werden. Dieser Editor läuft nur auf DOS-kompatiblen Personalcomputern. Die Dokumentation befindet sich in der Datei **ted.ps** im Verzeichnis **doc**.

- Das Verzeichnis **base**: Vordefinierter Prolog-Kode, der für die erstgenannten vier Komponenten notwendig ist.

Die erstgenannten vier Werkzeuge sind in [Dunker 93] genauer erklärt (die Datei `doc/tools.ps` im POSTSCRIPT-Format). Der Editor für Termersetzungssysteme (TED) ist in [Weisweber 94b] beschrieben (die Datei `doc/ted.ps` im POSTSCRIPT-Format).

Die Daten für die konzeptuelle Repräsentation (ATL) sind durch die Schnittstelle zum BACK-System festgelegt und sollten deshalb nicht verändert werden. Für die Daten im Verzeichnis `tbox` werden keine zusätzlichen Werkzeuge benötigt, da das BACK-System seine Daten in einem lesbaren Format einliest (Syntax siehe [Hoppe et al. 93]).

2.2.4 Dokumentationen

Das Verzeichnis `doc` enthält Dokumentationen für das MÜ-System und die Werkzeuge. Die entsprechenden Dateien enthalten POSTSCRIPT-Format. Der Inhalt der einzelnen Dateien ist folgendermaßen:

- Die Datei `rep100.ps`: Benutzerhandbuch für das BACK-System (identisch mit [Hoppe et al. 93])
- Die Datei `rep101.ps`: kurze Beschreibung des Algorithmus, der für die Interpretation der im MÜ-System verwendeten Termersetzungssysteme verwendet wird (identisch mit [Weisweber 92]).
- Die Datei `rep102.ps`: Beschreibung einer Erweiterung des Parsers für die direkte Interpretation von Metaregeln (identisch mit [Weisweber/Preuß 92]).
- Die Datei `iab28.ps`: Grundlagen für das Anapherninterpretationsverfahren (identisch mit [Dunker/Umbach 93])
- Die Datei `tools.ps`: Benutzungshandbuch für die GPSG-, Lexikon und FAS-Compiler (identisch mit [Dunker 93])
- Die Datei `ted.ps`: Benutzungshandbuch für den Editor für Termersetzungssysteme (TED, identisch mit [Weisweber 94b]).
- Die Datei `parser.ps`: Beschreibung des Algorithmus, der für den Parser im MÜ-System verwendet wird (identisch mit [Weisweber 87b])

2.3 Prolog-Programme

In diesem Abschnitt stelle ich die Prolog-Programme für die Unifikation und für den Interpretierer nicht-konfluenter Termersetzungssysteme (TES) vor. Sie werden in leicht modifizierter Form im MÜ-System des Projekts KIT-FAST eingesetzt.

Im Text und in den Kommentaren der Programme bezeichnet die Notation `pred/n` ein Prädikat mit Namen `pred`, das die Stelligkeit `n` hat. Außerdem verwende ich bei den Kommentaren folgendes Schema:

1. Schema für das dem Kommentar folgende Prädikat mit folgenden Argumentmodi:
 - *+Argument* für Argumente, die in einer Anfrage mit dem beschriebenen Prädikat instanziiert sein müssen (Eingabeargumente).

- *–Argument* für Argumente, die in einer Anfrage mit dem beschriebenen Prädikat variabel sein können, d.h. entweder instanziiert oder variabel sind (Ausgabeargumente). Bei Erfüllbarkeit einer Anfrage mit dem Prädikat wird *genau ein* Ergebnis erzeugt (deterministisch).
- *?Argument* für Argumente, die in einer Anfrage mit dem beschriebenen Prädikat variabel sein können, d.h. entweder instanziiert oder variabel sind (Ausgabeargumente). Bei Erfüllbarkeit einer Anfrage mit dem Prädikat kann *mehr als ein* Ergebnis erzeugt werden (nicht deterministisch).

2. Interpretation der einzelnen Argumente

3. Interpretation des Prädikats

4. Erläuterungen zu den einzelnen Klauseln des Prädikats.

2.3.1 Unifikation

Der Algorithmus für die Unifikation in diesem Abschnitt stammt in seinen Grundzügen aus [Eisele/Dörre 86]. Für das bessere Verständnis wird hier die Möglichkeit der Beschreibung von Pfaden in den TE-Regeln nicht berücksichtigt. Die beiden Argumente des Prädikats `unify_term/2` sind Terme, die Bäume darstellen und aus der Signatur `tree` in Abbildung 14 auf Seite 50 erzeugt werden. Der Algorithmus für die Unifikation geht davon aus, daß die Reihenfolge der Unterbäume auf den linken Seiten der TE-Regeln beliebig ist (zweites Argument von `unify_term/2`). Dies erhöht die Ausdruckstärke der TE-Regeln und wird durch das Prädikat `insert_term/3` berücksichtigt, dessen Semantik ähnlich der von `member/2` ist. Es unterscheidet sich nur dadurch, daß ein zusätzliches Argument die Liste ohne das gesuchte Element darstellt und daß die Elemente der Listen Bäume sind, aus denen ein Baum mit dem gegebenen Baum (1. Argument) unifizierbar sein muß. Die Aufgabe von `insert_term/3` liegt darin, aus der zweiten zu unifizierenden Liste von Bäumen denjenigen herauszusuchen, der mit einem Baum der ersten Liste unifizierbar ist. Anschließend werden die Reste der beiden Listen unifiziert.

```

/*****
/*** unify_term(+Tree1, +Tree2)                               ***/
/***                                                         ***/
/*** Tree1 ist ein Term gemaess der Signatur TREE.           ***/
/*** Tree2 ist ein Term gemaess der Signatur TREE.           ***/
/***                                                         ***/
/*** Unifiziert zwei Terme, die Baeume repraesentieren.       ***/
/*****/

```

```

unify_term(tree(Cat,List1), tree(Cat,List2)) :-
    unify_term_list(List1, List2).

```

```

/*****
/*** unify_term_list(+List1, +List2)                           ***/
/***                                                         ***/
/*** List1 ist eine Liste von Baeumen gemaess der Signatur TREE. ***/
/*** List2 ist eine Liste von Baeumen gemaess der Signatur TREE. ***/
/***                                                         ***/
/*** Unifiziert List1 und List2 und beruecksichtigt dabei die ***/
/*** Reihenfolgeunabhaengigkeit der Elemente in List2.       ***/
/***                                                         ***/

```

```

/** 1. Klausel: Liste1 oder Liste2 ist variabel.          **/
/** 2. Klausel: Liste1 und Liste2 sind leer.             **/
/** 3. Klausel: Der erste Baum von Liste1 wird mit einem Baum aus **/
/**               Liste2 unifiziert.                     **/
/*****                                                    *****/

unify_term_list(List1, List2) :-
    (var(List1); var(List2)),
    !,
    List1 = List2.

unify_term_list([], []).

unify_term_list([Tree|Rest1], List) :-
    insert_term(Tree, List, Rest2),
    unify_term_list(Rest1, Rest2).

/*****                                                    *****/
/** insert_term(?Tree, +Liste1, ?Liste2)                  **/
/**                                                    **/
/** Tree ist ein Term gemaess der Signatur TREE.         **/
/** Liste1 ist eine Eingabeliste von Baeumen gemaess der Signatur **/
/**               TREE.                                   **/
/** Liste2 ist eine Ausgabeliste von Baeumen gemaess der Signatur **/
/**               TREE.                                   **/
/**                                                    **/
/** Unifiziert Tree mit einem Element von Liste1, falls moeglich. **/
/** Liste2 ist dann Liste1 ohne Tree.                     **/
/**                                                    **/
/** 1. Klausel: Die Eingabeliste ist variabel.            **/
/** 2. Klausel: Der Baum ist mit dem ersten Element der Eingabe- **/
/**               liste unifizierbar. Der Cut darf hier nur im Fall **/
/**               der Eindeutigkeit der Unifikation verwendet werden. **/
/** 3. Klausel: Andere Loesungen fuer die Unifikation des Ein- **/
/**               gabebaums, falls die Eingabeliste nicht **/
/**               variabel und bisher kein anderer Baum der Eingabe- **/
/**               liste mit Tree1 unifizierbar war.        **/
/*****                                                    *****/

insert_term(Tree, List, [Tree|Rest]) :-
    var(List),
    !,
    List = [Tree|Rest].

insert_term(Tree1, [Tree2|Rest], Rest) :-
    unify_term(Tree1, Tree2),
    !.

insert_term(Tree1, [Tree2|Rest1], [Tree2|Rest2]) :-
    insert_term(Tree1, Rest1, Rest2).

```

Die drei *Cuts*, die in diesem Prolog-Programm verwendet werden sind *rote Cuts*, d.h. sie dürfen im Gegensatz zu *grünen Cuts* nicht weggelassen werden, da sonst die Semantik des Programms verändert würde (vgl. [Sterling/Shapiro 86]).

Bei diesem Algorithmus für die Unifikation muß beachtet werden, daß das *Cut* in der zweiten Klausel des Prädikats `insert_term/3` verhindert, daß die Unifikation von

Bäumen aufgrund der Reihenfolgeunabhängigkeit der Unterbäume auf den linken Seiten der TE-Regeln mehrdeutig ist. Fehlte dieser *Cut*, wäre zum Beispiel das Prädikat `unify_term(tree(a, [b,c]), tree(a, [X,Y]))` doppelt erfüllbar. Eine Lösung ist $\{X \leftarrow b, Y \leftarrow c\}$ und die andere $\{X \leftarrow c, Y \leftarrow b\}$. Solche Mehrfachlösungen wären dann durch Backtracking möglich. An dieser Stelle kann nur dann ein *Cut* verwendet werden, wenn auch bei beliebiger Reihenfolge der Unterbäume auf den linken Seiten der TE-Regeln die Eindeutigkeit der Unifikation garantiert werden kann. Dies erhöht die Effizienz des angegebenen Verfahrens für die Unifikation. Diese Anforderung wird durch die TE-Regeln im MÜ-System des Projekts KIT-FAST erfüllt.

2.3.2 Der Interpretierer für TES

In diesem Abschnitt wird das Prolog-Programm des Interpretierers für nicht-konfluente TES angegeben. Das erste Argument des Prädikats `rewrite_term/3`, das aus einen Eingabeterm einen äquivalenten Zielterm ableitet, ist die geordnete Menge von TE-Regelnummern. Sie wird als Liste repräsentiert, die folgende Elemente enthalten kann:

- Die Äquivalenzklassen der Relation $<_{app}$ (Zyklen der Anwendungsreihenfolge der TE-Regeln) sind Elemente der Form `cycle(Äquivalenzklasse)`. Die Äquivalenzklasse wird als Liste dargestellt.
- Die Ketten der Relation $<_{def}$ (Spezifität der linken Seiten der TE-Regeln) sind Elemente der Form `chain(Kette)`. Die Kette wird als Liste dargestellt.
- Die Äquivalenzklassen der Relation $\vee_{alt}$ (Mengen von verzweigenden TE-Regeln) sind Elemente der Form `alternatives(Äquivalenzklasse)`. Die Äquivalenzklasse wird als Liste dargestellt.

Die hier genannten drei Steuerungsrelationen werden in Abschnitt 1.2.3 skizziert. Eine ausführliche Beschreibung befindet sich in [Weisweber 94a] bzw. [Weisweber 92].

Das zweite und dritte Argument von `rewrite_term/3` sind der Eingabe- bzw. Zielterm, die Bäume darstellen und aus der Signatur `tree` in Abbildung 14 auf Seite 50 erzeugt werden. Die genaue Interpretation der Argumente ist Abschnitt 2.4.3 zu entnehmen.

Der Interpretierer wendet die TE-Regeln in der Reihenfolge an, die als erstes Argument von `rewrite_term/3` spezifiziert ist. Zyklen werden dabei durch das Prädikat `apply_cycle/3` solange angewendet, bis keine Regel aus dem Zyklus mehr anwendbar ist. Vor der Anwendung einer Regel durch das Prädikat `apply_rule/3` wird geprüft, ob es sich um eine Kette von Defaultregeln, verzweigende TE-Regeln oder eine "normale" TE-Regel handelt. Wenn eine Kette vorliegt, wird nur die erste anwendbare Regel angewendet. Handelt es sich um verzweigende TE-Regeln, werden alle Regeln alternativ angewendet. Die Alternativen entstehen durch Backtracking. Alle anderen Regeln werden durch das Prädikat `mult_apply_rule/3` solange auf den Eingabeterm angewendet, bis die Regel nicht mehr anwendbar ist.

Die TE-Regeln werden durch die Prädikate `rewrite_rule/3` als Fakten dargestellt (siehe Abschnitt 2.4.3 auf Seite 49). Das erste Argument ist die Nummer der TE-Regel. Das zweite Argument hat die Form *linke Seite* $\rightarrow$ *rechte Seite* und das dritte Argument ist die zusätzliche Bedingung für die linke Seite einer TE-Regel.

```

/*****/
/***/

```

```

/**      Interpretierer fuer nicht-konfluente TES      **/
/**      **/
/** Version 8.92      Wilhelm Weisweber **/
/*****/

/*****/
/** rewrite_term(+Liste, +Tree1, ?Tree2)      **/
/**      **/
/** Liste ist die Anwendungsreihenfolge der TE-Regeln mit folgenden **/
/**      Elementen:      **/
/**      - Nummern von TE-Regeln      **/
/**      - cycle(+Liste) (Zyklen der Anwendungsreihenfolge)      **/
/**      - chain(+Liste) (Ketten von Defaultregeln)      **/
/**      - alternatives(+Liste) (verzweigende TE-Regeln)      **/
/** Tree1 ist der Eingabeterm gemaess der Signatur TREE.      **/
/** Tree2 ist der Zielterm gemaess der Signatur TREE.      **/
/**      **/
/** Leitet Tree2 aus Tree1 ab, indem die TE-Regeln in der Reihen- **/
/** folge von Liste auf den Eingabeterm angewendet werden.      **/
/**      **/
/** 1. Klausel: Liste ist leer.      **/
/** 2. Klausel: Das Element von Liste ist ein Zyklus.      **/
/** 3. Klausel: Andere Elemente von Liste.      **/
/*****/

rewrite_term([], Tree, Tree).

rewrite_term([cycle(Cycle)|Rest], Tree, NewTree) :-
    !,
    apply_cycle(Cycle, Tree, IntTree),
    rewrite_term(Rest, IntTree, NewTree).

rewrite_term([Number|Rest], Tree, NewTree) :-
    apply_rule(Number, Tree, IntTree),
    rewrite_term(Rest, IntTree, NewTree).

/*****/
/** apply_cycle(+Liste, +Tree1, ?Tree2)      **/
/**      **/
/** Liste ist ein Zyklus der Anwendungsreihenfolge und Nummern von **/
/**      TE-Regeln als Elemente.      **/
/** Tree1 ist der Eingabeterm gemaess der Signatur TREE.      **/
/** Tree2 ist der Zielterm gemaess der Signatur TREE.      **/
/**      **/
/** Leitet Tree2 aus Tree1 ab, indem die TE-Regeln in der Reihen- **/
/** folge von Liste auf den Eingabeterm angewendet werden. Dabei **/
/** werden die TE-Regeln des Zyklus solange auf Tree1 angewendet, **/
/** bis keine Regel mehr anwendbar ist.      **/
/**      **/
/** 1. Klausel: Eine Regel aus Liste ist auf Tree1 anwendbar und es **/
/**      werden saemtliche auf diese TE-Regel folgenden      **/
/**      Regeln des Zyklus ueberprueft.      **/
/** 2. Klausel: Keine Regel aus Liste ist auf Tree1 anwendbar.      **/
/*****/

apply_cycle(Cycle, Tree, NewTree) :-
    select(Number, Cycle, Rest),

```

```

    superpose(Tree, Number),
    !,
    rewrite_term([Number|Rest], Tree, IntTree),
    apply_cycle(Cycle, IntTree, NewTree).

apply_cycle(_, Tree, Tree).

/*****
/**** apply_rule(+Element, +Tree1, ?Tree2) ****/
/****
/**** Element ist die Nummer einer TE-Regel, eine Kette von Default- ****/
/****          regeln oder eine Menge von verzweigenden TE-Regeln. ****/
/**** Tree1   ist der Eingabeterm gemaess der Signatur TREE. ****/
/**** Tree2   ist der Zielterm gemaess der Signatur TREE. ****/
/****
/**** Leitet Tree2 aus Tree1 ab, indem entweder die spezifischste ****/
/**** TE-Regel aus einer Kette von Defaultregeln, eine Menge von ****/
/**** verzweigenden TE-Regeln oder eine TE-Regel auf Tree1 angewendet ****/
/**** wird. ****/
/****
/**** 1. Klausel: Element ist eine Kette, deren erstes Element eine ****/
/****          Menge von verzweigenden TE-Regeln ist, die alle an- ****/
/****          wendbar sind. Nur diese Regeln der Kette werden an- ****/
/****          gewendet. Es entstehen durch Backtracking ueber ****/
/****          apply_rule/3 so viele Loesungen wie es Elemente in ****/
/****          der Menge der verzweigenden TE-Regeln gibt. ****/
/****
*****/

apply_rule(chain([alternatives([Number|Rest])|_]), Tree, NewTree) :-
    superpose(Tree, Number),
    !,
    apply_rule(alternatives([Number|Rest]), Tree, NewTree).

/*****
/**** 2. Klausel: Element ist eine Kette, deren erste TE-Regel ****/
/****          anwendbar ist. Nur diese Regel wird angewendet. ****/
/****
*****/

apply_rule(chain([Number|_]), Tree, NewTree) :-
    superpose(Tree, Number),
    !,
    mult_apply_rule(Number, Tree, NewTree).

/*****
/**** 3. Klausel: Element ist eine Kette, deren erste TE-Regel nicht ****/
/****          anwendbar ist. In diesem Fall wird die naechste ****/
/****          TE-Regel der Kette auf Anwendung ueberprueft. ****/
/****
*****/

apply_rule(chain([_|Rest]), Tree, NewTree) :-
    !,
    apply_rule(chain(Rest), Tree, NewTree).

/*****
/**** 4. Klausel: Element ist eine Kette, die entweder leer war, oder ****/
/****          aus der keine TE-Regel anwendbar war. In diesem ****/
/****          Fall wird die Kette uebersprungen. ****/
/****
*****/

```

```

apply_rule(chain([]), Tree, Tree) :-
    !.

/*****/
/**** 5. Klausel: Element ist eine Menge von verzweigenden TE-Regeln ****/
/****          und die erste TE-Regel ist anwendbar. Damit sind ****/
/****          alle verzweigenden TE-Regeln anwendbar und werden ****/
/****          alternativ durch Backtracking ueber member/2 ange- ****/
/****          wendet. Es entstehen so viele Verzweigungen wie es ****/
/****          Elemente in der Menge der verzweigenden TE-Regeln ****/
/****          gibt. ****/
/*****/

apply_rule(alternatives([Number1|Rest]), Tree, NewTree) :-
    superpose(Tree, Number1),
    !,
    member(Number2, [Number1|Rest]),
    mult_apply_rule(Number2, Tree, NewTree).

/*****/
/**** 6. Klausel: Element ist eine Menge von verzweigenden TE-Regeln ****/
/****          geln und die erste TE-Regel ist nicht anwendbar. In ****/
/****          diesem Fall sind alle anderen TE-Regeln der Menge ****/
/****          ebenfalls nicht anwendbar und die Alternativen ****/
/****          werden uebergangen. ****/
/*****/

apply_rule(alternatives(_), Tree, Tree) :-
    !.

/*****/
/**** 7. Klausel: Element ist die Nummer einer TE-Regel. Diese wird ****/
/****          auf Anwendbarkeit ueberprueft und ggf. mehrfach auf ****/
/****          den Eingabeterm Tree angewendet. ****/
/*****/

apply_rule(Number, Tree, NewTree) :-
    mult_apply_rule(Number, Tree, NewTree).

/*****/
/**** mult_apply_rule(+Number, +Tree1, ?Tree2) ****/
/**** ****/
/**** Number ist die Nummer einer TE-Regel. ****/
/**** Tree1 ist der Eingabeterm gemaess der Signatur TREE. ****/
/**** Tree2 ist der Zielterm gemaess der Signatur TREE. ****/
/**** ****/
/**** Leitet Tree2 aus Tree1 ab, indem die TE-Regel mit der Nummer ****/
/**** solange auf Tree1 angewendet wird, bis dies nicht mehr moeglich ****/
/**** ist und ueberprueft nach jeder Anwendung den Bedingungsteil. ****/
/**** ****/
/**** 1. Klausel: Die TE-Regel ist anwendbar. ****/
/**** 2. Klausel: Die TE-Regel ist nicht (mehr) anwendbar. ****/
/*****/

mult_apply_rule(Number, Tree, NewTree) :-
    rewrite_rule(Number, Left --> Right, Conds),
    rewrite_term(Tree, Left, Right, IntTree),
    check_conditions(Conds), % hier nicht weiter definiert
    !,
    mult_apply_rule(Number, IntTree, NewTree).

```

```
mult_apply_rule(_, Tree, Tree).
```

```

/*****/
/** superpose(+Tree, +Number) ****/
/** ****/
/** Tree ist ein Term gemaess der Signatur TREE. ****/
/** Number ist die Nummer einer TE-Regel. ****/
/** ****/
/** Ueberlagert Tree mit der linken Seite der TE-Regel mit der ****/
/** Nummer. ****/
/*****/

```

```

superpose(Tree, Number) :-
    rewrite_rule(Number, Left --> _),
    rewrite_term(Tree, Left, _, _),
    !.

```

```

/*****/
/** rewrite_term(+Tree1, +Tree2, +Tree3, ?Tree4) ****/
/** ****/
/** Tree1 ist der Eingabeterm gemaess der Signatur TREE. ****/
/** Tree2 ist der Term auf der linken Seite einer TE-Regel gemaess ****/
/** der Signatur TREE. ****/
/** Tree3 ist der Term auf der rechten Seite einer TE-Regel gemaess ****/
/** der Signatur TREE. ****/
/** Tree4 ist der Zielterm gemaess der Signatur TREE. ****/
/** ****/
/** Wendet eine TE-Regel mit der linken Seite Tree2 und der rechten ****/
/** Seite Tree3 auf Tree1 an, indem Tree1 mit Tree2 ueberlagert ****/
/** wird. Wenn das moeglich ist wird der entsprechende Unterterm ****/
/** von Tree1 durch Tree3 ersetzt. ****/
/** ****/
/** 1. Klausel: Tree ist mit der linken Seite unifizierbar und wird ****/
/** durch die rechte Seite Right ersetzt. ****/
/** 2. Klausel: Das 1. Argument ist nicht mit der linken Seite uni- ****/
/** fizierbar. In diesem Fall wird versucht, einen der ****/
/** Unterbaeume mit der linken Seite zu ueberlagern. ****/
/*****/

```

```

rewrite_term(Tree, Left, Right, Right) :-
    unify_term(Tree, Left),
    !.

```

```

rewrite_term(tree(Cat, List), Left, Right, tree(Cat, NewList)) :-
    rewrite_terms(List, Left, Right, NewList).

```

```

/*****/
/** rewrite_terms(+List1, +Tree1, +Tree2, ?Liste2) ****/
/** ****/
/** List1 ist eine Liste von Baeumen des Eingabeterms gemaess der ****/
/** Signatur TREE. ****/
/** Tree1 ist der Term auf der linken Seite einer TE-Regel gemaess ****/
/** der Signatur TREE. ****/
/** Tree2 ist der Term auf der rechten Seite einer TE-Regel ****/
/** gemaess der Signatur TREE. ****/
/** Liste2 ist eine Liste von Baeumen des Zielterms gemaess der ****/
/** Signatur TREE. ****/

```

```

/**/
/**/ Wendet auf einen der Baeume in Liste1 eine TE-Regel mit der
/**/ linken Seite Tree1 und der rechten Seite Tree2 an und ersetzt
/**/ diesen Baum in Liste2 durch den abgeleiteten.
/**/
/**/ 1. Klausel: Der erste Baum von Liste1 ist mit der linken Seite
/**/               der TE-Regel ueberlagerbar und Tree wird durch
/**/               NewTree ersetzt.
/**/ 2. Klausel: Der erste Baum von Liste1 ist nicht mit der linken
/**/               Seite der TE-Regel ueberlagerbar. In diesem Fall
/**/               wird versucht, den naechsten Baum der Liste mit der
/**/               linken Seite zu ueberlagern.
/**/
/*****

rewrite_terms([Tree|List], Left, Right, [NewTree|List]) :-
    rewrite_term(Tree, Left, Right, NewTree),
    !.

rewrite_terms([Tree|List], Left, Right, [Tree|NewList]) :-
    rewrite_terms(List, Left, Right, NewList).

/*****
/**/ select(?Element, +Liste1, ?Liste2)
/**/
/**/ Element ist irgendein Term.
/**/ Liste1 ist eine Liste mit beliebigen Elementen.
/**/ Liste2 ist eine Liste mit beliebigen Elementen.
/**/
/**/ Ueberprueft, ob Element in Liste1 vorkommt, oder liefert ein
/**/ Element von Liste1. Liste2 ist der Teil von Liste1, der nach
/**/ Element folgt.
/**/
/**/ 1. Klausel: Das erste Element von Liste1 ist das gesuchte.
/**/ 2. Klausel: Das erste Element ist nicht das gesuchte. In diesem
/**/               Fall wird das naechste Element geprueft.
/**/
/*****

select(Element, [Element|Rest], Rest).

select(Element, [_|Rest], List) :-
    select(Element, Rest, List).

```

2.4 Die interne Darstellung der linguistischen Daten

In diesem Abschnitt werden die linguistischen Daten des MÜ-Systems für die Übersetzungsrichtung Deutsch → Englisch vorgestellt. Dazu werden die *Argumentschemata* $\text{prädikat}(\text{Argument}_1, \dots, \text{Argument}_n)$ der *Prädikate* für Prolog-Fakten angegeben. Dabei wird der Modus jedes Arguments spezifiziert mit

- $+\text{Argumentname}$ für Argumente, die in einer Anfrage mit dem Prädikat instanziiert sein müssen (Eingabeargumente),
- $-\text{Argumentname}$ für Argumente, die in einer Anfrage mit dem Prädikat variabel sein können, d.h. entweder instanziiert oder variabel sind (Ausgabeargumente). Bei Erfüllbarkeit wird genau ein Ergebnis erzeugt (deterministisch) sowie

$\text{cat} =$
sorts: CAT, LANG, $F_1, \dots, F_n$
opns: C: LANG $F_1 \dots F_m \rightarrow \text{CAT}$ mit $m \leq n$
 $\text{gpsg, hyb, g, atl, e} \rightarrow \text{LANG}$
 $V_{i_1}, \dots, V_{i_k} \rightarrow F_i$ für $1 \leq i \leq n$

Abbildung 13: Signaturschema für komplexe Kategorien

- *?Argumentname* für Argumente, die in einer Anfrage mit dem Prädikat variabel sein können, d.h. entweder instanziiert oder variabel sind (Ausgabeargumente). Bei Erfüllbarkeit wird mehr als ein Ergebnis erzeugt (nicht deterministisch).

Zunächst werden dazu diejenigen Daten erläutert, die von mehr als einem Übersetzungsschritt benötigt werden. Dies sind Daten zur Definition von komplexen Kategorien, Termersetzungssystemen (TES) und kontextfreien Grammatiken. Anschließend wird die interne Darstellung der linguistischen Daten für die morphologische Analyse und Synthese, für die syntaktische Analyse, für die semantische Analyse und die Generierung, für die konzeptuelle Analyse, für die Anapherninterpretation, für den Transfer und für das Wissensrepräsentationssystem BACK beschrieben. Dazu werden die einzelnen Dateien, die für den jeweiligen Übersetzungsschritt geladen werden müssen, durch Kürzel identifiziert. Mithilfe dieser Kürzel und des Prädikats `gram_file/5`, das in der Datei `mtmenu.pl` definiert ist (siehe Abschnitt 3.5.1 und Anhang B), können die entsprechenden Dateinamen ermittelt werden. Auf diese Weise können dem MÜ-System auch Dateien mit selbst definierten linguistischen Daten zur Verfügung gestellt werden.

2.4.1 Komplexe Kategorien

Unter einer komplexen Kategorie verstehe ich eine (atomare) Kategorie (Hauptkategorie), der eine Menge von Merkmal-Wert-Paaren zugeordnet ist. Prinzipiell kann eine komplexe Kategorie auch nur aus einer Menge von Merkmal-Wert-Paaren bestehen. Dieser Fall kann allerdings simuliert werden, indem für eine Sprache nur eine einzige Hauptkategorie verwendet wird (siehe z.B. GPSG-Kategorien im Abschnitt 2.4.6).

In Abbildung 13 wird das Signaturschema für komplexe Kategorien angegeben. Für jede Kategorie C wird ein Operator definiert. Diese Operatoren werden im folgenden als Hauptkategorie bezeichnet. Jede Hauptkategorie besitzt eine festgelegte Anzahl von $n + 1$ Argumentsorten LANG, $F_1, \dots, F_n$, die Merkmale repräsentieren. Darunter ist die ausgezeichnete Sorte LANG, welche die Zugehörigkeit der entsprechenden Kategorie zu einem bestimmten Vokabular kennzeichnet. Dieses Merkmal wird benötigt, um die Terminierung der TES zu gewährleisten. Die Werte eines Merkmals werden als Konstante definiert. Alle Konstanten einer Sorte ergeben den Wertebereich des entsprechenden Merkmals. Als Beispiel für den Wertebereich des durch die Sorte LANG repräsentierten Merkmals sind hier die im Projekt KIT-FAST verwendeten Vokabulare genannt.

In Prolog können komplexe Kategorien als Terme realisiert werden. Der Prädikatname des Terms entspricht der Hauptkategorie und die Argumente des Prädikats entsprechen den Werten der Merkmale, die der Hauptkategorie zugeordnet sind.

GPSG-Kategorien

Kategorien der Generalisierten Phrasenstrukturgrammatiken (GPSG) werden auf zwei verschiedene Arten intern repräsentiert. Einerseits werden sie in der syntaktischen Analyse benötigt (siehe Abschnitt 1.3.2) und andererseits in der semantischen Analyse und der Generierung, die beide durch Termersetzungssysteme realisiert sind (siehe Abschnitt 1.3.3 bzw. 1.3.6).

In der syntaktischen Analyse wird nur eine Hauptkategorie `cat` verwendet. Die Stelligkeit des entsprechenden Prolog-Prädikats ist identisch mit der Anzahl der für die entsprechende Sprache definierten Merkmale. Die GPSG-Kategorien für die syntaktische Analyse sind in den Dateien definiert, die in der Datei `mtmenu.pl` das Kürzel `id` haben (siehe Abschnitt 3.5.1 bzw. Anhang B). Zur Definition werden die Prolog-Prädikate `category_position/2`, `possible_values/2` und `card_features/2` verwendet (siehe Abschnitt 2.4.6).

In der Regel erfordern natürliche Sprachen ein Vielzahl von Merkmalen, die alle in jeder GPSG-Kategorie vorkommen müssen. Für eine Kategorie sind aber normalerweise nicht alle Merkmale relevant, z.B. sind Nomen nicht für Tempus spezifiziert. Außerdem würde die Ausgabe von syntaktischen Strukturen ziemlich unübersichtlich werden. Deshalb wird mithilfe des Prolog-Prädikates `alias/2` (siehe Abschnitt 2.4.6) eine Abbildung definiert, welche die vollständigen GPSG-Kategorien in abgekürzte Varianten (Alias) abbildet, die aus einer Hauptkategorie wie z.B. `s`, `np`, `vp`, `pp` usw. bestehen und nur noch die Werte der relevanten Merkmale enthalten. Diese Abbildung wird in den Dateien definiert, die in der Datei `mtmenu.pl` das Kürzel `alias` haben (siehe Abschnitt 3.5.1 bzw. Anhang B).

Die GPSG-Kategorien werden nach Abschluß der syntaktischen Analyse in entsprechende Terme abgebildet. Die syntaktischen Strukturen werden dann als äquivalente Terme in der Datenbasis abgespeichert (siehe Abschnitt 3.3.3). Deshalb wird mithilfe des Prolog-Prädikates `map_alias/2` (siehe Abschnitt 2.4.7) eine Abbildung definiert, welche die vollständigen GPSG-Kategorien in entsprechende Kategorienterme abbildet. Diese Abbildung wird in den Dateien definiert, die in der Datei `mtmenu.pl` das Kürzel `gdef` haben (siehe Abschnitt 3.5.1 bzw. Anhang B).

In der semantischen Analyse und der Generierung, die durch Termersetzungssysteme realisiert sind, sehen die *GPSG-Kategorienterme* den Alias-Definitionen sehr ähnlich. Sie unterscheiden sich im wesentlichen dadurch, daß sie ein zusätzliches erstes Argument haben, das die Sprache spezifiziert, für welche die entsprechende Kategorie definiert ist (Wert des Merkmals `lang(uage)`, das für sämtliche Kategorienterme definiert ist). Mithilfe dieses Merkmals wird die Termination der Termersetzungssysteme gewährleistet. Die GPSG-Kategorienterme sind in den Dateien definiert, die in der Datei `mtmenu.pl` das Kürzel `gdef` haben (siehe Abschnitt 3.5.1 bzw. Anhang B). Zur Definition werden die Prolog-Prädikate `category_def/2` und `feature_def/2` verwendet (siehe Abschnitt 2.4.2).

FAS-Kategorien

Kategorien der Funktor-Argument-Struktur (FAS) werden in der semantischen und konzeptuellen Analyse, im Transfer sowie in der Generierung verwendet (siehe Abschnitte 1.3.3 bis 1.3.6). Diese Komponenten werden durch Termersetzungssysteme realisiert, so daß es sich hier um *FAS-Kategorienterme* handelt. Es gibt mehrere FAS-Hauptkategorien wie z.B. `fas`, `illoc`, `clausefin`, `clause`, `term` usw. (siehe Abschnitt 1.1.3). Das erste Argument jeder FAS-Kategorie spezifiziert die Sprache, für die sie definiert ist (Wert des Merkmals `lang(uage)`). Mithilfe dieses Merkmals wird die Ter-

mination der Termersetzungssysteme gewährleistet. Die FAS-Kategorienterme sind in den Dateien definiert, die in der Datei `mtmenu.pl` das Kürzel `fdef` haben (siehe Abschnitt 3.5.1 bzw. Anhang B). Zur Definition werden die Prolog-Prädikate `category_def/2` und `feature_def/2` verwendet (siehe Abschnitt 2.4.2).

ATL-Kategorien

Kategorien der ABox-Tell-Language (ATL) des BACK-Systems werden in der konzeptuellen Analyse verwendet (siehe Abschnitt 1.3.4), die durch ein Termersetzungssystem realisiert ist, so daß es sich hier um *ATL-Kategorienterme* handelt. Es gibt drei ATL-Hauptkategorien `atl`, `equal`, `with` (siehe Abschnitt 1.1.4). Das erste Argument jeder ATL-Kategorie spezifiziert die Sprache, für die sie definiert ist (Wert des Merkmals `lang(uage)`). Mithilfe dieses Merkmals wird die Termination des Termersetzungssystems gewährleistet. Die ATL-Kategorienterme sind in den Dateien definiert, die in der Datei `mtmenu.pl` das Kürzel `adef` haben (siehe Abschnitt 3.5.1 bzw. Anhang B). Zur Definition werden die Prolog-Prädikate `category_def/2` und `feature_def/2` verwendet (siehe Abschnitt 2.4.2).

2.4.2 Daten für komplexe Kategorien

Die Definition von komplexen Kategorien besteht aus der Zuordnung von Merkmalen zu einer Hauptkategorie und der Festlegung des Wertebereichs für jedes Merkmal.

`category_def(?Bezeichner, ?Hauptkategorie, ?Merkmale)`

1. *Bezeichner* ist ein Atom (Bezeichner für eine Repräsentationsebene) und zwar
 - (a) `gpsg`: GPSG-Kategorienterme des Deutschen bzw. syntaktische Kategorienterme des Englischen,
 - (b) `hyb`: hybride Kategorienterme (mit syntaktischen und semantischen Informationen) des Deutschen bzw. Englischen,
 - (c) `g`: FAS-Kategorienterme des Deutschen,
 - (d) `atl`: ATL-Kategorienterme des Deutschen und
 - (e) `e`: FAS-Kategorienterme des Englischen.
2. *Hauptkategorie* ist ein Atom (Prädikat eines Terms für eine *komplexe Kategorie*).
3. *Merkmale* ist eine Liste von Atomen (Merkmalen).

Mit dem Prädikat `category_def/3` werden die Merkmale, die einer Hauptkategorie einer bestimmten Repräsentationsebene zugeordnet sind, definiert.

`feature_def(?Bezeichner, ?Merkmal, ?Wertebereich)`

1. *Bezeichner* ist ein Atom (Bezeichner für eine Repräsentationsebene). Mögliche Werte sind unter `category_def/3` (s.o.) nachzulesen.
2. *Merkmal* ist ein Atom.

3. *Wertebereich* ist eine Liste mit Atomen oder ganzen Zahlen (Wertebereich des Merkmals).

Mit dem Prädikat **feature_def**/ β werden die Wertebereiche der Merkmale von komplexen Kategorien einer bestimmten Repräsentationsebene definiert.

2.4.3 Daten für Termersetzungssysteme

Das Termersetzungsverfahren (siehe Abschnitt 1.2.3) ist der Kern des MÜ-Systems. Die Termersetzungssysteme (TES), die von diesem Verfahren auf die Strukturen der verschiedenen Repräsentationsebenen angewendet werden, bestehen aus mehreren Teilen:

- Ersetzungsrichtung (Ausgangsstruktur und Zielstruktur),
- Reihenfolgerelation $<_{app}$ (siehe Abschnitt 1.2.3),
- Verzweigungsrelation $\vee_{alt}$ (siehe Abschnitt 1.2.3),
- Defaultrelation $<_{def}$ (siehe Abschnitt 1.2.3),
- Anwendungsreihenfolge sämtlicher Termersetzungsregeln,
- für jede Termersetzungsregel: Menge der komplexen Kategorien der Ausgangsstruktur, die auf der linken Seite der TE-Regel vorkommen, und
- Termersetzungsregeln (TE-Regeln).

direction($-Ausgangsstruktur \dashrightarrow -Zielstruktur$)

1. *Ausgangsstruktur* ist ein Atom (Ausgangsstrukturbezeichner).
2. *Zielstruktur* ist ein Atom (Zielstrukturbezeichner).

Das Prädikat **direction**/ 1 definiert die Ausgangs- und Zielstruktur, für ein TES.

prec($?Nummer1, ?Nummer2$)

1. *Nummer1* ist eine natürliche Zahl (Nummer einer TE-Regel).
2. *Nummer2* ist eine natürliche Zahl (Nummer einer TE-Regel).

Das Prädikat **prec**/ 2 definiert die Relation $<_{app}$ (siehe Abschnitt 1.2.3) und spezifiziert, daß die TE-Regel mit der Nummer *Nummer1* vor der TE-Regel mit der Nummer *Nummer2* angewendet werden bzw. auf Anwendung geprüft werden muß.

alt(?Nummer1, ?Nummer2, ?Nummer3, ?Nummer4)

1. *Nummer1* ist eine natürliche Zahl (Nummer einer TE-Regel).
2. *Nummer2* ist eine natürliche Zahl (Nummer einer TE-Regel).
3. *Nummer3* ist eine natürliche Zahl (Nummer einer TE-Regel).
4. *Nummer4* ist eine natürliche Zahl (Nummer einer TE-Regel).

Das Prädikat **alt**/4 definiert die Relation $\vee_{alt}$ (siehe Abschnitt 1.2.3) und spezifiziert, daß die beiden TE-Regeln mit den Nummern *Nummer3* und *Nummer4* verzweigende TE-Regeln sind, die aus der Überlagerung der linken Seiten der TE-Regeln mit den Nummern *Nummer1* und *Nummer2* berechnet wurden.

default(?Nummer1, ?Nummer2)

1. *Nummer1* ist eine natürliche Zahl (Nummer einer TE-Regel).
2. *Nummer2* ist eine natürliche Zahl (Nummer einer TE-Regel).

Das Prädikat **default**/2 definiert die Relation $<_{def}$ (siehe Abschnitt 1.2.3) und spezifiziert, daß die linke Seite der TE-Regel mit der Nummer *Nummer1* spezifischer ist als die linke Seite der TE-Regel mit der Nummer *Nummer2*. Die TE-Regel mit der Nummer *Nummer1* muß vor der TE-Regel mit der Nummer *Nummer2* auf Anwendung überprüft werden. Sobald eine TE-Regel einer Kette der Relation $<_{def}$ anwendbar ist, wird sie angewendet. Sämtliche nachfolgende Regeln der Kette mit allgemeineren linken Seiten, die in diesem Fall ebenfalls anwendbar wären, werden nicht mehr angewendet.

ordered_prec_set(–*Liste*)

1. *Liste* ist eine Liste, die aus folgenden Elementen besteht:
 - Nummern von TE-Regeln oder
 - Listen mit Nummern von TE-Regeln (Äquivalenzklasse der Relation $<_{cyc}$, siehe Abschnitt 1.2.3).

Das Prädikat **ordered_prec_set**/1 definiert die Reihenfolge der Anwendung von TE-Regeln. Die Liste ist nach den Relationen $<_{app}$ und $<_{def}$ (siehe Abschnitt 1.2.3) sortiert. Ist das Element eine Nummer, so wird die entsprechende TE-Regel solange angewendet, bis sie nicht mehr anwendbar ist. Wenn das Element eine Liste mit Nummern ist, werden sämtliche TE-Regeln mit den entsprechenden Nummern solange angewendet, bis keine TE-Regel aus der Liste mehr anwendbar ist.

rw_cats(?Nummer, ?Kategorienliste, ?Bedingung)

1. *Nummer* ist eine natürliche Zahl (Nummer einer TE-Regel).
2. *Kategorienliste* ist eine Liste mit Kategorientermen der Ausgangsstruktur, die auf der linken Seite der entsprechenden TE-Regel vorkommen.

3. *Bedingung* ist eine Liste von Bedingungen für die Anwendung der entsprechenden TE-Regel. Die Liste wird als Konjunktion von Bedingungen interpretiert. Die einzelnen Bedingungen können folgendermaßen aussehen:

- *+Term1 rel +Term2*
Dies ist eine Bedingung über Grundterme, das sind Merkmalwerte, Kategorienterme, Bäumen oder Listen von Bäumen mit und ohne Variablenvorkommen. Dabei ist *rel* $\in \{=, \neq, <, \geq\}$.
- *(+SUB, ?CAS) crn (+CONF, ?ROLE)*
Eine solche Bedingung greift auf das Prädikat *cr_noun/4* der Kasus-Rollen-Tabelle zu (siehe Abschnitt 2.4.7), wobei *SUB* und *CONF* die syntaktische bzw. semantische Subkategorisierung eines Nomens sind. *CAS* und *ROLE* sind der Oberflächen- bzw. Tiefenkasus (semantische Rolle) eines seiner Argumente.
- *(+SUB, ?CAS) crv (+CONF, ?ROLE)*
Eine solche Bedingung wird analog zum Prädikat *crn* für Verben interpretiert. Es greift auf das Prädikat *cr_verb/4* der Kasus-Rollen-Tabelle zu (siehe Abschnitt 2.4.7).
- *(+SUB, -THEM, +MC) crl (+CONF, -THEM1, +Liste)*
Eine solche Bedingung benutzt das Prädikat *cr_verb/4* der Kasus-Rollen-Tabelle für jedes Element aus *Liste*, welche die Argumente eines Verbs enthält. Dabei werden die thematischen Positionen (FAS-Merkmal *them* bzw. hybrides Merkmal *them1*) eines Verbs und seiner Argumente ermittelt. Die thematische Position des Verbs hängt vom GPSG-Merkmal *mc* ab, das angibt, ob Verbzweit- oder Verbedstellung vorliegt.
- *(+Merkmal, +Kategorie) val -Wert*
Eine solche Bedingung ermittelt den Wert eines Merkmals einer komplexen Kategorie.
- *(+Liste1, +Liste2) app -Liste3*
Eine solche Bedingung konkateniert *Liste1* und *Liste2* zu *Liste3*.
- *+Bedingung1 or +Bedingung2*
Eine solche Bedingung gilt, wenn eine der beiden Bedingungen erfüllt ist.

Mithilfe des Prädikats *rw_cats/3* wird festgestellt, welche TE-Regeln eines TES auf einen Eingabeterm anwendbar sind. Eine TE-Regel ist möglicherweise anwendbar, wenn die Kategorien der Ausgangsstruktur auf ihrer linken Seite im Eingabeterm vorkommen. Die Menge der auf Anwendbarkeit zu prüfenden TE-Regeln reduziert sich dadurch erheblich.

rewrite_rule(?Nummer, ?LS --> ?RS, ?Bedingungen)

1. *Nummer* ist eine natürliche Zahl (Nummer der TE-Regel).
2. *LS* ist ein Term (linke Seite der TE-Regel), der durch die Spezifikation *tree* in Abbildung 14 erzeugt wird. Die Spezifikation *cat* für komplexe Kategorien findet sich in Abbildung 13 auf Seite 44.
3. *RS* ist ein Term (rechte Seite einer TE-Regel), der durch die Spezifikation *tree* in Abbildung 14 erzeugt wird.

```

tree = cat +
  sorts: TREE, LIST
  opns: tree: CAT LIST → TREE
        list: TREE LIST → LIST
        []: → LIST

```

Abbildung 14: Signatur für Ableitungsbäume

4. *Bedingungen* siehe `rw_cats/3`.

Mithilfe des Prädikats `rewrite_rule/3` werden die TE-Regeln für ein TES definiert.

2.4.4 Daten für kontextfreie Grammatiken mit komplexen Kategorien

Kontextfreie Grammatiken mit komplexen Kategorien (siehe Abschnitt 2.4.2) werden für die syntaktische Überprüfung von Strukturen benötigt, auf die keine TE-Regeln mehr anwendbar sind. Solche Strukturen heißen *minimal* oder *irreduzibel*. Eine syntaktische Kontrolle ist erforderlich, weil TE-Regeln eines TES unvollständig oder fehlerhaft sein können, so daß nicht garantiert ist, daß eine irreduzible Struktur nach der Anwendung der TE-Regeln auch der Zielsyntax entspricht. Zur Definition einer *kontextfreien Grammatik* gehören *Metaregeln*, *kontextfreie Regeln* und ein *Lexikon*. Die Metaregeln, die hier verwendet werden, unterscheiden sich von denen in der syntaktischen Analyse (vgl. Abschnitt 2.4.6), da sie nur eine Kategorie löschen dürfen.

`meta(?Kategorie1, ?Kategorie2, ?Kategorie3, ?Kategorie1')`

1. *Kategorie1* ist ein Term (komplexe Mutterkategorie einer kontextfreien Regel, die Bedingung für die Anwendung der Metaregel ist).
2. *Kategorie2* ist ein Term (komplexe Tochterkategorie einer kontextfreien Regel, die Bedingung für die Anwendung der Metaregel ist).
3. *Kategorie3* ist ein Term (komplexe Tochterkategorie einer kontextfreien Regel, die gelöscht werden soll).
4. *Kategorie1'* ist ein Term (komplexe Mutterkategorie einer kontextfreien Regel, durch die *Kategorie1* ersetzt werden soll, wenn die Metaregel auf die kontextfreie Regel anwendbar ist).

Mit dem Prädikat `meta/4` werden die *Metaregeln* definiert, die auf die kontextfreien Regeln angewendet werden. Eine Metaregel kann nur eine Kategorie löschen.

`?Nummer : ?Kategorie ==> ?Kategorienliste`

1. *Nummer* ist eine natürliche Zahl (Nummer einer kontextfreien Regel).
2. *Kategorie* ist ein Term (komplexe Mutterkategorie).

3. *Kategorienliste* ist eine Liste mit komplexen Kategorientermen (Tochterkategorien). Eine komplexe Kategorie in eckigen Klammern ($[?Kategorie]$) ist optional und eine komplexe Kategorie als Argument des Prädikats $\mathbf{s}/1$ ($\mathbf{s}(?Kategorie)$) kann null- oder mehrfach vorkommen.

Mit dem Prädikat $\mathbf{==>}/2$ werden *kontextfreie Regeln* mit Mutter *Kategorie* und Töchtern *Kategorienliste* definiert. Die Definition der Daten für komplexe Kategorien ist im Abschnitt 2.4.2 nachzulesen.

$\mathbf{flex}(?Terminal, ?Kategorie, ?Zuordnung)$

1. *Terminal* ist ein Atom (terminales Symbol).
2. *Kategorie* ist ein Term (komplexe Kategorie).
3. *Zuordnung* ist ein Atom (Zuordnung für das terminale Symbol).

Das Prädikat $\mathbf{flex}/3$ wird zur Definition eines *Lexikons* für kontextfreie Grammatiken verwendet. Wird es für *FAS-Lexika* verwendet, so ist das dritte Argument ein *TBox-Konzept*. Auf diese Weise wird das *Übergangsllexikon* $FAS \rightarrow TBox$ definiert. Für alle anderen Lexika ist das dritte Argument ohne Bedeutung (meist eine anonyme Variable).

2.4.5 Daten für die morphologische Analyse bzw. Synthese

Die Komponenten für die morphologische Analyse und Synthese basieren auf dem SUTRA-System. Die Prädikate, mit denen die morphologischen Daten definiert werden, sind hier nur kurz beschrieben. Eine Kurzbeschreibung des Verfahrens findet sich im Abschnitt 1.2.1. Eine ausführliche Dokumentation enthalten [Busemann 83a] und [Busemann 83b]. Im einzelnen werden folgende Komponenten benötigt:

- GPSG-Lexikon (Übergangsllexikon $GPSG \leftrightarrow FAS$),
- FAS-Lexikon (Übergangsllexikon $FAS \leftrightarrow TBox$),
- morphologisches Lexikon (Wortstamm $\leftrightarrow$ morphologische Klasse) und
- Flexionslexikon (morphologische Klasse $\leftrightarrow$ Morphem).

Daten in Dateien mit dem Kürzel $\mathbf{glex}$

$\mathbf{glex}(?GPSG-Terminal, ?GPSG-Kategorie, ?FAS-Terminal, ?FAS-Kategorie)$

1. *GPSG-Terminal* ist ein Atom (terminales Symbol der GPSG, Wortstamm).
2. *GPSG-Kategorie* ist ein Term mit dem Prädikat $\mathbf{cat}$ (GPSG-Kategorie).
3. *FAS-Terminal* ist ein Atom (terminales Symbol der FAS).
4. *FAS-Kategorie* ist ein Term (komplexe FAS-Kategorie).

Mithilfe des Prädikats $\mathbf{glex}/4$ wird das *Übergangsllexikon* $GPSG \rightarrow FAS$ definiert.

Daten in Dateien mit dem Kürzel **flex**

Hier handelt es sich um das Übergangslexikon $FAS \rightarrow TBox$. Das Prädikat **flex/3** ist im Abschnitt 2.4.4 beschrieben. Dieses Lexikon wird in der morphologischen Analyse benötigt, um sämtliche (semantischen) Lesarten für die semantische Analyse zur Verfügung zu haben. Um einen zweifachen Zugriff auf das Übergangslexikon $GPSG \rightarrow FAS$ in der syntaktischen *und* semantischen Analyse zu vermeiden, werden die lexikalischen Übergänge $GPSG \rightarrow FAS$ bereits beim Lexikonzugriff während der syntaktischen Analyse durchgeführt.

Daten in Dateien mit dem Kürzel **mlex**

mlex(*?Attribut*, *?GPSG-Terminal*, *?Wert*)

1. *Attribut* ist ein Atom (morphologisches Attribut).
2. *GPSG-Terminal* ist ein Atom (terminales Symbol einer GPSG, Wortstamm eines Nomens oder Verbs).
3. *Wert* ist ein Atom oder eine ganze Zahl (Wert des Attributs).

Mit dem Prädikat **mlex/3** werden morphologische Attribute für terminale GPSG-Symbole (Wortstämme) spezifiziert. Für *Nomen des Deutschen* können folgende Attribute und Werte vorkommen:

- **ldeklin** (Deklinationsklasse) mit den Werten **n_s**, **en_s**, **e**, **z**, **s** oder **zz**.
- **lumlaut** (Umlautung im Plural) mit den Werten **yes**, falls der Wortstamm im Plural umgelautet wird, oder **no**, falls der Wortstamm im Plural nicht umgelautet wird.
- **lnplu** (unregelmäßige Pluralbildung) mit der unregelmäßigen Form als Wert.

Für *Verben des Deutschen* können folgende Attribute und Werte vorkommen:

- **lpaux** (Hilfsverb für das Partizip Perfekt) mit den Werten **hab** oder **sein**.
- **lpprf** (Partizip Perfekt und Passiv möglich) mit dem Partizip als Wert.
- **lprfp** (kein Partizip Passiv möglich) mit der Form im Partizip Perfekt als Wert.
- **lvtyp** (Konjugationsklasse) mit den Werten **1** bis **11**.
- **lvgfa** (Grundform eines Verbs des Segments **a**) mit der entsprechenden Grundform als Wert. Für Verben des Segments **a** sind Stammform und Grundform identisch.
- **lvgfb** (Grundform eines Verbs des Segments **b**) mit der Grundform als Wert.
- **lvgfc** (Grundform eines Verbs des Segments **c**) mit der Grundform als Wert.
- **lvgfd** (Grundform eines Verbs des Segments **d**) mit der Grundform als Wert.
- **lvgfs** (Grundform eines Verbs des Segments **s**) mit der Grundform als Wert. Dieses Segment stellt unregelmäßige Grundformen dar.
- **lvrbz** (abtrennbares Verbpräfix) mit einem entsprechenden Präfix als Wert.

Für *Nomen des Englischen* können folgende Attribute und Werte vorkommen:

- **ntyp** (Deklinationsklasse) mit den Werten 1 bis 9.
- **plu** (unregelmäßige Pluralbildung) mit der unregelmäßigen Form als Wert.

Für *Verben des Englischen* können folgende Attribute und Werte vorkommen:

- **vtyp** (Konjugationsklasse) mit den Werten 1 bis 16.
- **pret** (unregelmäßiges Präteritum) mit der unregelmäßigen Form als Wert.
- **past** (unregelmäßiges Partizip Perfekt) mit der unregelmäßigen Form als Wert.
- **3sg** (unregelmäßige Verbform in der 3. Person Singular) mit der unregelmäßigen Verbform als Wert.
- **ger** (unregelmäßiges Gerundium) mit der unregelmäßigen Form als Wert.

Daten in Dateien mit dem Kürzel **mdata**

detrel(*?PLU*, *?KAS*, *?GEN*, *?DET/REL*, *?MORPH*)

1. *PLU* (Plural) mit den Werten **+** oder **-**.
2. *KAS* (Kasus) mit den Werten **nom**, **gen**, **dat** oder **acc**.
3. *GEN* (Genus) mit den Werten 1 (maskulinum), 2 (femininum) oder 3 (neutrum).
4. *DET/REL* mit den Werten **det** (Artikel) oder **rel** (Relativpronomen).
5. *MORPH* mit dem entsprechenden Morphem als Wert.

Mit dem Prädikat **detrel/5** wird die Relation zwischen syntaktischen Merkmalen und Morphemen für *Artikel* und *Relativpronomen* definiert.

possflex(*?PLU*, *?KAS*, *?GEN*, *?MORPH*)

1. *PLU* (Plural) mit den Werten **+** oder **-**.
2. *KAS* (Kasus) mit den Werten **nom**, **gen**, **dat** oder **acc**.
3. *GEN* (Genus) mit den Werten 1 (maskulinum), 2 (femininum) oder 3 (neutrum).
4. *MORPH* mit dem entsprechenden Morphem als Wert.

Mit dem Prädikat **possflex/4** wird die Relation zwischen syntaktischen Merkmalen und Morphemen für *Possessivpronomen* definiert.

quant(?PLU, ?GEN, ?KAS, ?FLEX, ?MORPH)

1. *PLU* (Plural) mit den Werten + oder -.
2. *GEN* (Genus) mit den Werten 1 (maskulinum), 2 (femininum) oder 3 (neutrum).
3. *KAS* (Kasus) mit den Werten **nom**, **gen**, **dat** oder **acc**.
4. *FLEX* (Flexionsart) mit den Werten 1 (Artikel stark, Adjektiv schwach), 2 (Artikel schwach, Adjektiv stark) oder 3 (Artikel gemischt, Adjektiv gemischt).
5. *MORPH* mit dem entsprechenden Morphem als Wert.

Mit dem Prädikat **quant/5** wird die Relation zwischen syntaktischen Merkmalen und Morphemen für *Quantoren* definiert.

adj(?PLU, ?GEN, ?KAS, ?FLEX, ?MORPH)

1. *PLU* (Plural) mit den Werten + oder -.
2. *GEN* (Genus) mit den Werten 1 (maskulinum), 2 (femininum) oder 3 (neutrum).
3. *KAS* (Kasus) mit den Werten **nom**, **gen**, **dat** oder **acc**.
4. *FLEX* (Flexionsart) mit den Werten 1 (Artikel stark, Adjektiv schwach), 2 (Artikel schwach, Adjektiv stark) oder 3 (Artikel gemischt, Adjektiv gemischt).
5. *MORPH* mit dem entsprechenden Morphem als Wert.

Mit dem Prädikat **adj/5** wird die Relation zwischen syntaktischen Merkmalen und Morphemen für *Adjektive* definiert.

noun(?DEKLIN, ?PLU, ?KAS, ?GEN, ?MORPH)

1. *DEKLIN* (Deklinationsklasse) mit den Werten **n_s**, **en_s**, **e**, **z**, **s** oder **zz**.
2. *PLU* (Plural) mit den Werten + oder -.
3. *KAS* (Kasus) mit den Werten **nom**, **gen**, **dat** oder **acc**.
4. *GEN* (Genus) mit den Werten 1 (maskulinum), 2 (femininum) oder 3 (neutrum).
5. *MORPH* mit dem entsprechenden Morphem als Wert.

Mit dem Prädikat **noun/5** wird die Relation zwischen syntaktischen Merkmalen und Morphemen für *Nomen* definiert.

prn(?PLU, ?PER, ?KAS, ?GEN, ?MORPH)

1. *PLU* (Plural) mit den Werten **+** oder **-**.
2. *PER* (Person) mit den Werten 1, 2 oder 3.
3. *KAS* (Kasus) mit den Werten **nom**, **gen**, **dat** oder **acc**.
4. *GEN* (Genus) mit den Werten 1 (maskulinum), 2 (femininum) oder 3 (neutrum).
5. *MORPH* mit dem entsprechenden Morphem als Wert.

Mit dem Prädikat **prn**/5 wird die Relation zwischen syntaktischen Merkmalen und Morphemen für *Personalpronomen* definiert.

refl(?PLU, ?PER, ?GEN, ?KAS, ?MORPH)

1. *PLU* (Plural) mit den Werten **+** oder **-**.
2. *PER* (Person) mit den Werten 1, 2 oder 3.
3. *GEN* (Genus) mit den Werten 1 (maskulinum), 2 (femininum) oder 3 (neutrum).
4. *KAS* (Kasus) mit den Werten **nom**, **gen**, **dat** oder **acc**.
5. *MORPH* mit dem entsprechenden Morphem als Wert.

Mit dem Prädikat **refl**/5 wird die Relation zwischen syntaktischen Merkmalen und Morphemen für *Reflexivpronomen* definiert.

vseg(?TEMP/MOD, ?PLU, ?PER, ?TYP, ?SEGMENT)

1. *TEMP/MOD* (Tempus/Modus) mit den Werten **pres** (Präsens), **impf** (Imperfekt), **kon2** (Konjunktiv II), **imp** (Imperativ), **kon1** (Konjunktiv I), **inf** (Infinitiv) oder **par1** (Partizip Präsens).
2. *PLU* (Plural) mit den Werten **+** oder **-**.
3. *PER* (Person) mit den Werten 1, 2 oder 3.
4. *TYP* (Konjugationsklasse) mit den Werten 1 bis 11.
5. *SEGMENT* mit den Werten **a**, **b**, **c**, **d** oder **s**.

Mit dem Prädikat **vseg**/5 wird die Relation zwischen syntaktischen Merkmalen und Segmenten für *Verben* definiert.

verb(?SEGMENT, ?TEMP/MOD, ?PLU, ?PER, ?MORPH)

1. *SEGMENT* mit den Werten **a**, **b**, **c**, **d** oder **s**.
2. *TEMP/MOD* (Tempus/Modus) mit den Werten **pres** (Präsens), **impf** (Imperfekt), **kon2** (Konjunktiv II), **imp** (Imperativ), **kon1** (Konjunktiv I), **inf** (Infinitiv) oder **par1** (Partizip Präsens).
3. *PLU* (Plural) mit den Werten **+** oder **-**.
4. *PER* (Person) mit den Werten **1**, **2** oder **3**.
5. *MORPH* mit dem entsprechenden Morphem als Wert.

Mit dem Prädikat **verb/5** wird die Relation zwischen Segmenten und Morphemen für *Verben* definiert.

2.4.6 Daten für den Parser

Die syntaktische Analyse ist neben der morphologischen Analyse und Synthese die einzige Komponente, die eine Strukturabbildung ohne das Termersetzungsverfahren durchführt. Sie wird durch einen GPSG-Parser mit direkter Interpretation von Metaregeln und ID/LP-Regeln realisiert (siehe Abschnitt 1.2.2 für eine Kurzbeschreibung des Parsers und [Weisweber 87a], [Weisweber 87b], [Weisweber 88] und [Weisweber/Preuß 92] für Details). Die Daten, die der Parser benötigt, sind

- Abkürzungen für GPSG-Kategorien (Alias),
- Feature-Co-occurrence-Restrictions (FCR),
- Metaregeln,
- Immediate-Dominance-Regeln (ID-Regeln),
- Kommentare zu den ID-Regeln,
- Dominanzrelation für die ID-Regeln,
- Linear-Precedence-Aussagen (LP-Aussagen),
- Definitionen für GPSG-Kategorien (vgl. Abschnitt 2.4.1),
- FOOT-Merkmale und
- Spezifikationsmerkmale (HEAD- und Kongruenz-Merkmale).

Daten in Dateien mit dem Kürzel **alias**

alias(-Alias, +GPSG-Kategorie)

1. *Alias* ist ein Term (Kurzform einer GPSG-Kategorie).
2. *GPSG-Kategorie* ist ein Term mit dem Prädikat **cat** (GPSG-Kategorie).

Mit dem Prädikat **alias/2** werden Abkürzungen für interne GPSG-Kategorien (*Alias*) definiert, um sie leichter lesbar zu machen.

Daten in Dateien mit dem Kürzel fcr

`fcr(?Nummer, ?Bedingung, ?Konsequenz)`

1. *Nummer* ist eine natürliche Zahl (Nummer einer Feature-Co-occurrence-Restriction, FCR).
2. *Bedingung* ist eine Liste mit Paaren der Form [*?Merkmal*, *?Wert*], welche die Bedingung für die Anwendung einer FCR spezifiziert. Dabei ist
 - (a) *Merkmal* ist ein Atom (GPSG-Merkmal).
 - (b) *Wert* ist ein Atom oder eine ganze Zahl (Wert des GPSG-Merkmals).
3. *Konsequenz* ist eine Liste mit Paaren der Form [*?Merkmal*, *?Wert*] (Konsequenz der Anwendung einer FCR). *Merkmal* und *Wert* siehe oben bei *Bedingung*

Mit dem Prädikat `fcr/3` werden die *Feature-Co-occurrence-Restrictions* (*FCRs*) einer GPSG definiert.

Daten in Dateien mit dem Kürzel id

`meta(?GPSG-Kat1, ?GPSG-Kat2, ?Liste, ?GPSG-Kat2', ?GPSG-Kat1')`

1. *GPSG-Kat1* ist ein Term mit dem Prädikat `cat` (GPSG-Tochterkategorie einer ID-Regel, die Bedingung für die Anwendung der Metaregel ist). Der Term kann auch `nil` sein. Dadurch wird keine GPSG-Tochterkategorie als Bedingung spezifiziert.
2. *GPSG-Kat2* ist ein Term mit dem Prädikat `cat` (GPSG-Mutterkategorie einer ID-Regel, die Bedingung für die Anwendung der Metaregel ist).
3. *Liste* ist eine Liste mit GPSG-Tochterkategorientermen einer ID-Regel, die Bedingung für die Anwendung der Metaregel ist.
4. *GPSG-Kat2'* ist ein Term mit dem Prädikat `cat` (GPSG-Mutterkategorie, durch die *GPSG-Kat2* ersetzt werden soll, wenn die Metaregel auf eine ID-Regel anwendbar ist).
5. *GPSG-Kat1'* ist ein Term mit dem Prädikat `cat` (GPSG-Tochterkategorie, durch die *GPSG-Kat1* ersetzt werden soll, wenn die Metaregel auf eine ID-Regel anwendbar ist). Ist *GPSG-Kat1* `nil`, so wird *GPSG-Kat1'* als Tochterkategorie zur ID-Regel hinzugefügt. Ist *GPSG-Kat1'* `nil`, so wird *GPSG-Kat1* gelöscht, falls diese nicht auch `nil` ist.

Mit dem Prädikat `meta/5` werden die *Metaregeln* einer GPSG definiert.

`id(?Nummer, ?Mutter, ?Töchter)`

1. *Nummer* ist eine natürliche Zahl (Nummer einer ID-Regel).
2. *Mutter* ist ein Term (GPSG-Mutterkategorie der ID-Regel).
3. *Töchter* ist ein Term der Form `r(?Obligatorisch, ?Optional, ?Stern)`, wobei
 - (a) *Obligatorisch* eine Liste mit den obligatorischen GPSG-Tochterkategorien ist,

- (b) *Optional* eine Liste mit den optionalen GPSG-Tochterkategorien ist und
- (c) *Stern* eine Liste mit den GPSG-Tochterkategorien ist, die null- oder mehrfach vorkommen dürfen.

Mit dem Prädikat `id/3` werden die *Immediate-Dominance-Regeln* (*ID-Regeln*) einer GPSG definiert.

`last_rule(id, -Nummer)`

1. *Nummer* ist eine natürliche Zahl (Anzahl der ID-Regeln).

Mit dem Prädikat `last_rule/2` werden die Anzahl der Regeln einer GPSG-Komponente festgehalten.

`category_position(?Merkmal, ?Position)`

1. *Merkmal* ist ein Atom (Name eines Merkmals).
2. *Position* ist eine natürliche Zahl (Argumentposition des Werts für das entsprechende Merkmal in einer GPSG-Kategorie).

Mit dem Prädikat `category_position/2` werden die Positionen der Werte für die definierten Merkmale einer GPSG-Kategorie definiert (vgl. Abschnitt 2.4.1).

`possible_values(?Merkmal, ?Wertebereich)`

1. *Merkmal* ist ein Atom (Name eines Merkmals).
2. *Wertebereich* ist eine Liste mit Atomen oder ganzen Zahlen (Wertebereich eines Merkmals). Der Merkmalwert '^' (undefiniert) ist für jedes Merkmal erlaubt und muß nicht im Wertebereich angegeben werden.

Mit dem Prädikat `possible_values/2` werden die Wertebereiche der Merkmale von GPSG-Kategorien festgelegt (vgl. Abschnitt 2.4.1).

`card_features(-Stelligkeit)`

1. *Stelligkeit* ist eine natürliche Zahl (Stelligkeit des Prädikats `cat`, mit dem GPSG-Kategorienterme repräsentiert werden).

Mit dem Prädikat `card_features/1` wird die Anzahl der Merkmale festgelegt, die für GPSG-Kategorien definiert sind (vgl. Abschnitt 2.4.1).

`foot_range(-FOOT-Merkmale)`

1. *FOOT-Merkmale* ist eine Liste von Atomen (Menge der FOOT-Merkmale).

Mit dem Prädikat `foot_range/1` wird die Menge der *FOOT-Merkmale* definiert.

`specification_feature(?Merkmal, ?Wert, ?Merkmalliste)`

1. *Merkmal* ist ein Atom (Spezifikationsmerkmal).
2. *Wert* ist ein Atom oder eine ganze Zahl (Wert des Spezifikationsmerkmals).
3. *Merkmalliste* ist eine Liste von Atomen (Menge von GPSG-Merkmalen).

Mit dem Prädikat `specification_feature/3` wird eine Menge von Merkmalen definiert, deren Werte in den Kategorien eines lokalen Baumes unifiziert werden, falls der Wert des *Spezifikationsmerkmals* in einer Kategorie mit dem angegebenen identisch ist (siehe [Preuß 89]). Dadurch wird das *Agreement-Prinzip* (AP, siehe Abschnitt 1.1.2) realisiert.

`head_spec(?Merkmal, ?Wert, ?Merkmalliste)`

1. *Merkmal* ist ein Atom (GPSG-Merkmal).
2. *Wert* ist ein Atom oder eine ganze Zahl (Wert des GPSG-Merkmals).
3. *Merkmalliste* ist eine Liste von Atomen (Menge von GPSG-Merkmalen).

Mit dem Prädikat `head_spec/3` wird eine Menge von Merkmalen definiert, deren Werte in den Kategorien eines lokalen Baumes unifiziert werden, falls der Wert des angegebenen Merkmals in einer Kategorie mit dem angegebenen identisch ist. Eine solche Tochterkategorie ist in etwa der *Head* eines lokalen Baumes und die Identifikation der Merkmalwerte realisiert die *Head-Feature-Convention* (HFC, siehe Abschnitt 1.1.2). Die HFC unterscheidet sich vom AP nur dadurch, daß bei der Unifikation von Merkmalwerten in einem lokalen Baum ebenfalls die Tiefenkategorien der Mutter mit berücksichtigt werden.

Daten in Dateien mit dem Kürzel `idc`

`id_comment(?Nummer, ?Kommentar)`

1. *Nummer* ist eine natürliche Zahl (Nummer einer ID-Regel).
2. *Kommentar* ist ein Atom (Kommentar zu der ID-Regel, meist in Hochkommata).

Mit dem Prädikat `id_comment/2` können Kommentare zu ID-Regeln definiert werden, die bei bestimmten Tracing-Routinen (siehe Abschnitt 3.6) ausgegeben werden. In der Regel ist der Kommentar die Kurzform einer ID-Regel.

Daten in Dateien mit dem Kürzel `lp`

`trans_lp(?GPSG-Kategorie1, ?GPSG-Kategorie2)`

1. *GPSG-Kategorie1* ist ein Term mit dem Prädikat `cat` (GPSG-Kategorie).
2. *GPSG-Kategorie2* ist ein Term mit dem Prädikat `cat` (GPSG-Kategorie).

Mit dem Prädikat `trans_lp/2` werden die *Linear-Precedence-Aussagen* (LP-Aussagen) einer GPSG definiert. Eine solche LP-Aussage definiert, daß *GPSG-Kategorie1* als Tochter in jedem lokalen Baum vor der Schwester *GPSG-Kategorie2* vorkommen muß.

Daten in Dateien mit dem Kürzel dom

`dom(?GPSG-Kategorie, ?Nummer)`

1. *GPSG-Kategorie* ist ein Term mit dem Prädikat `cat` (GPSG-Kategorie).
2. *Nummer* ist eine natürliche Zahl (Nummer einer ID-Regel).

Mit dem Prädikat `dom/2` wird die *Dominanzrelation* für den Parser definiert (siehe [Weisweber 87a] und [Weisweber 87b]). Sie sagt aus, daß *GPSG-Kategorie* in der ID-Regel mit der Nummer *Nummer* dominiert wird, d.h. als Tochter vorkommt.

2.4.7 Daten für die semantische Analyse bzw. Generierung

Die semantische Analyse und die Generierung werden durch entsprechende Termersetzungssysteme realisiert (siehe Abschnitt 1.3.3 bzw. 1.3.6). Folgende Komponenten müssen zur Verfügung stehen:

- Abbildung von GPSG-Kategorien in GPSG-Kategorienterme,
- Definition der GPSG-Kategorienterme für Ausgangs- bzw. Zielsprache,
- Definition der hybriden Kategorienterme für Ausgangs- bzw. Zielsprache,
- Definition der FAS-Kategorienterme für Ausgangs- bzw. Zielsprache,
- FAS-Grammatik der Ausgangssprache bzw. Grammatik der Zielsprache,
- FAS-Lexikon der Ausgangssprache bzw. Lexikon der Zielsprache,
- Kasus-Rollen-Tabellen für Ausgangs- bzw. Zielsprache (Oberflächenkasus $\leftrightarrow$ Tiefenkasus) und
- Termersetzungssystem für die semantische Analyse bzw. Generierung.

Daten in Dateien mit dem Kürzel gdef

Hier handelt es sich um die Definitionen von *GPSG-Kategorientermen* des Deutschen (vgl. Abschnitt 2.4.1). Die Prädikate `category_def/3` und `feature_def/3` sind im Abschnitt 2.4.2 beschrieben. Der Wert des Arguments *Bezeichner* ist `gpsg`.

Diese Datei enthält außerdem die Abbildung von GPSG-Kategorien (siehe Abschnitt 2.4.1) in GPSG-Kategorienterme, wie sie für die Termersetzung benötigt werden.

`map_alias(-GPSG-Terminal, +GPSG-Kategorie)`

1. *GPSG-Terminal* ist ein Term (GPSG-Kategorienterm).
2. *GPSG-Kategorie* ist ein Term mit dem Prädikat `cat` (GPSG-Kategorie).

Mit dem Prädikat `map_alias/2` wird die Abbildung von *GPSG-Kategorien* in *GPSG-Kategorienterme* definiert, die an der Schnittstelle zwischen dem GPSG-Parser und dem Termersetzungsverfahren benötigt wird.

Daten in Dateien mit dem Kürzel hdef

Hier handelt es sich um die Definitionen von *hybriden Kategorientermen* des Deutschen, die sowohl syntaktische als auch semantische Informationen enthalten. Die Prädikate `category_def/3` und `feature_def/3` sind im Abschnitt 2.4.2 beschrieben. Der Wert des Arguments *Bezeichner* ist *hyb*.

Daten in Dateien mit dem Kürzel fdef

Hier handelt es sich um die Definitionen von *FAS-Kategorientermen* des Deutschen (vgl. Abschnitt 2.4.1). Die Prädikate `category_def/3` und `feature_def/3` sind im Abschnitt 2.4.2 beschrieben. Der Wert des Arguments *Bezeichner* ist *g*.

Daten in Dateien mit dem Kürzel fas

Hier handelt es sich um kontextfreie *FAS-Regeln* des Deutschen. Das Prädikat `==>/2` ist im Abschnitt 2.4.4 beschrieben. Die nicht-terminalen Symbole sind FAS-Kategorienterme (vgl. Abschnitt 2.4.1).

Daten in Dateien mit dem Kürzel idgen

Hier handelt es sich um kontextfreie Regeln mit Metaregeln für die *Generierung* im Englischen. Die Prädikate `==>/2` und `meta/4` sind im Abschnitt 2.4.4 beschrieben. Die nicht-terminalen Symbole sind komplexe syntaktische Kategorienterme (vgl. Abschnitt 2.4.1).

Daten in Dateien mit dem Kürzel flex

Hier handelt es sich um das *Übergangslexikon* $FAS \rightarrow TBox$ des Deutschen. Das Prädikat `flex/3` ist im Abschnitt 2.4.4 beschrieben.

Daten in Dateien mit dem Kürzel conf

`cr_verb(?Sprache, ?SUBCAT, ?Kasusspezifikationen, ?CONF, ?Rollen)`

1. *Sprache* ist ein Atom (*g* oder *e*).
2. *SUBCAT* ist eine natürliche Zahl (Wert für das Merkmal SUBCAT einer GPSG, das die syntaktischen Subkategorisierung angibt, siehe Abschnitt 1.1.1 und 1.1.2).
3. *Kasusspezifikationen* ist eine Liste mit Paaren der Form *?Merkmal* : *?Wert*.
 - (a) *Merkmal* ist ein Atom (*cas* im Deutschen sowie *cas* und *pf* für Präposition im Englischen).
 - (b) *Wert* ist ein Atom (Oberflächenkasus),
 - im Deutschen: *nom*, *gen*, *dat*, *acc* und *pd* für Prädikative sowie
 - im Englischen
 - für das Merkmal *cas*: *nom*, *acc* und *clause* sowie
 - für das Merkmal *pf*: die englischen Präpositionen.
4. *CONF* ist ein Atom (Wert für das Merkmal CONF der FAS, das die semantische Konfiguration angibt, siehe Abschnitt 1.1.3).

5. *Rollen* ist eine Liste mit Atomen (semantische Rollen, die Werte des Merkmals `role` der FAS sind, siehe Abschnitt 1.1.3).

Mit dem Prädikat `cr_verb/5` werden *Kasus-Rollen-Tabellen* für Verben dargestellt. Sie definieren die Relationen zwischen Oberflächenkasus und Tiefenkasus (semantischen Rollen) bei den Argumenten von Verben. Die Liste mit den Kasusspezifikationen und die Liste mit den semantischen Rollen müssen gleich lang sein, denn das n -te Element der ersten Liste wird dem n -ten Element der zweiten Liste zugeordnet. Die Tabellen können für beide Richtungen verwendet werden, also sowohl für die semantische Analyse als auch für die Generierung.

`cr_noun(?Sprache, ?SUBCAT, ?Kasusspezifikationen, ?CONF, ?Rollen)`

Mit dem Prädikat `cr_noun/5` werden *Kasus-Rollen-Tabellen* für Nomen dargestellt. Die Interpretation der Argumente und die Verwendung des Prädikats ist identisch zu `cr_verb/5` (s.o.).

Daten in Dateien mit dem Kürzel `map`

Hier handelt es sich um das Termersetzungssystem für die *semantische Analyse*. Die Daten für Termersetzungssysteme sind im Abschnitt 2.4.3 beschrieben.

Daten in Dateien mit dem Kürzel `pa`

Hier handelt es sich um das Termersetzungssystem für die *Generierung*. Die Daten für Termersetzungssysteme sind im Abschnitt 2.4.3 beschrieben.

2.4.8 Daten für die konzeptuelle Analyse

Die konzeptuelle Analyse wird durch ein entsprechendes Termersetzungssystem realisiert (siehe Abschnitt 1.3.4). Folgende Komponenten müssen zur Verfügung stehen:

- Definition der FAS-Kategorienterme,
- Definition der ATL-Kategorienterme,
- ATL-Grammatik,
- ATL-Lexikon und
- Termersetzungssystem für die konzeptuelle Analyse.

Daten in Dateien mit dem Kürzel `fdef`

Hier handelt es sich um die Definitionen von *FAS-Kategorientermen* des Deutschen (vgl. Abschnitt 2.4.1). Die Prädikate `category_def/3` und `feature_def/3` sind im Abschnitt 2.4.2 beschrieben. Der Wert des Arguments *Bezeichner* ist `g`.

Daten in Dateien mit dem Kürzel `adef`

Hier handelt es sich um die Definitionen von *ATL-Kategorientermen* des Deutschen (vgl. Abschnitt 2.4.1). Die Prädikate `category_def/3` und `feature_def/3` sind im Abschnitt 2.4.2 beschrieben. Der Wert des Arguments *Bezeichner* ist `atl`.

Daten in Dateien mit dem Kürzel `atlgram`

Hier handelt es sich kontextfreie *ATL-Regeln* des Deutschen. Das Prädikat `==>/2` ist im Abschnitt 2.4.4 beschrieben. Die nicht-terminalen Symbole sind ATL-Kategorienterme (vgl. Abschnitt 2.4.1). Da die vordefinierte ATL-Grammatik durch das BACK-System (vgl. [Hoppe et al. 93]) vorgegeben ist, sollte sie nicht verändert werden.

Daten in Dateien mit dem Kürzel `atllex`

Hier handelt es sich um das *ATL-Lexikon*. Das Prädikat `flex/3` ist im Abschnitt 2.4.4 beschrieben. Das dritte Argument *Zuordnung* wird nicht benötigt und ist deshalb eine anonyme Variable.

Daten in Dateien mit dem Kürzel `fasat1`

Hier handelt es sich um das Termersetzungssystem für die *konzeptuelle Analyse*. Die Daten für Termersetzungssysteme sind im Abschnitt 2.4.3 beschrieben.

2.4.9 Daten für das Anapherninterpretationsverfahren

Die Prädikate, mit denen die Daten für das Anapherninterpretationsverfahren definiert werden, sind hier nur kurz beschrieben. Eine Kurzbeschreibung des Verfahrens findet sich im Abschnitt 1.2.4. Eine ausführliche Dokumentation enthält [Dunker/Umbach 93]. Im einzelnen werden folgende Komponenten benötigt:

- Pronomentabelle,
- Grenzwert,
- Abstandswert,
- Reihenfolge der Kriterien,
- Gewicht der Kriterien und
- Anwendungs- und Bewertungsmodi der Kriterien.

Daten in Dateien mit dem Kürzel `pront`

`pronspectab_g_5(?Pronomen, ?FAS-Kategorie)`

1. *Pronomen* ist ein Atom.
2. *FAS-Kategorie* ist ein Term mit einer FAS-Hauptkategorie (in der Regel das Atom `term`) als Prädikat. Die Argumente dieses Prädikats haben die Form `(?Merkmal, ?Wert)` mit
 - (a) *Merkmal* ist ein Atom (FAS-Merkmal).
 - (b) *Wert* ist ein Atom oder eine ganze Zahl (Wert des FAS-Merkmals).

Mit dem Prädikat `pronspectab_g_5/2` werden die Pronomen festgelegt, die durch das Anapherninterpretationsverfahren berücksichtigt werden sollen.

Daten in Dateien mit dem Kürzel `edefs`

`getGrenzwert(−Grenzwert)`

1. *Grenzwert* ist eine ganze Zahl.

Mit dem Prädikat `getGrenzwert/1` wird der Grenzwert für Bewertungen von Antezedenskandidaten definiert. Er bestimmt, wann ein Antezedenskandidat ausgeschlossen wird. Dies geschieht, wenn die Summe seiner Bewertungen echt kleiner als der Grenzwert ist. Ein sehr tiefer, negativer Wert ist meistens unwirksam.

`getAbstand(−Abstand)`

1. *Abstand* ist eine ganze Zahl.

Mit dem Prädikat `getAbstand/1` wird bestimmt, wann ein Antezedenskandidat ausscheidet. Dies ist der Fall, wenn die Summe seiner Bewertungen weiter, als der Abstandswert es zuläßt, hinter der des besten Kandidaten zurückliegt. Der Abstandswert wird unwirksam, wenn er sehr hoch gewählt wird.

`kriterien(−Kriterien)`

1. *Kriterien* ist eine Liste mit Atomen (Kriterien).

Mit dem Prädikat `kriterien/1` werden die *Kriterien* und die Reihenfolge ihrer Anwendung auf Antezedenskandidaten festgelegt. Mögliche Kriterien sind `kongruenz`, `p_bin-dungen`, `naehe`, `subjekt`, `satzthema`, `rolle`, `freie_ergaenzung` und `diskurs`.

`gewicht(?Kriterium, ?Gewicht)`

1. *Kriterium* ist ein Atom.
2. *Gewicht* ist eine ganze Zahl.

Mit dem Prädikat `gewicht/2` werden die Gewichte der einzelnen Kriterien festgelegt. Das Gewicht eines Kriteriums wird während der Auswertung mit einem aktuell ermittelten Faktor multipliziert. Das Ergebnis ist die Bewertung eines Antezedenskandidaten.

`krit_modes(−Auswertungsmodi)`

1. *Auswertungsmodi* ist eine Liste mit Termen, die den Namen eines Kriteriums als Prädikat haben. Jedes Prädikat ist 2-stellig und hat die Argumente
 - (a) *+Anwendungsmodus* mit den Atomen `strukt` oder `ref` als Wert und
 - (b) *+Bewertungsmodus* mit den Atomen `strukt` oder `ref` als Wert.

Mit dem Prädikat `krit_modes/1` werden der Anwendungs- und Bewertungsmodus jedes einzelnen Kriteriums festgelegt. Der Anwendungsmodus kann strukturell oder referentiell sein, d.h. die Kriterien erfassen strukturelle bzw. referentielle Eigenschaften des Antezedenskandidaten. Dies bedeutet, daß die Kriterien auf eine Antezedenslinie bzw. auf das Referenzobjekt einer Gruppe von Antezedenten angewendet werden.

Der Bewertungsmodus kann ebenfalls strukturell oder referentiell sein, d.h. die Kriterien bewerten die strukturellen Positionen der Antezedenskandidaten einzeln bzw. die gesamte Gruppe (das Referenzobjekt) gleichzeitig.

Die Kombination referentielle Anwendung und strukturelle Bewertung ist ausgeschlossen, weil bei einer referentiellen Anwendung keines der strukturellen Teile für eine Bewertung ausgezeichnet ist.

2.4.10 Daten für den Transfer

Der Transfer wird durch ein entsprechendes Termersetzungssystem realisiert (siehe Abschnitt 1.3.5). Folgende Komponenten müssen zur Verfügung stehen:

- Definition der FAS-Kategorienterme der Ausgangssprache,
- Definition der FAS-Kategorienterme der Zielsprache,
- FAS-Grammatik der Zielsprache,
- FAS-Lexikon der Zielsprache,
- Termersetzungssystem für den Transfer.

Daten in Dateien mit dem Kürzel `fdef`

Hier handelt es sich um die Definitionen von *FAS-Kategorientermen* des Deutschen und des Englischen (vgl. Abschnitt 2.4.1). Die Prädikate `category_def/3` und `feature_def/3` sind im Abschnitt 2.4.2 beschrieben. Der Wert des Arguments *Bezeichner* ist `g` bzw. `e`.

Daten in Dateien mit dem Kürzel `fas`

Hier handelt es sich um kontextfreie *FAS-Regeln* des Englischen. Das Prädikat `==>/2` ist im Abschnitt 2.4.4 beschrieben. Die nicht-terminalen Symbole sind FAS-Kategorienterme (vgl. Abschnitt 2.4.1).

Daten in Dateien mit dem Kürzel `flex`

Hier handelt es sich um das *Übergangsllexikon* $FAS \rightarrow TBox$ des Englischen. Das Prädikat `flex/3` ist im Abschnitt 2.4.4 beschrieben.

Daten in Dateien mit dem Kürzel `trule`

Hier handelt es sich um das Termersetzungssystem für den *Transfer*. Die Daten für Termersetzungssysteme sind im Abschnitt 2.4.3 beschrieben.

2.4.11 Daten für das BACK-System

Das terminologische Wissen, das in der TBox des BACK-Systems definiert wird, muß dem MÜ-System vor der Verarbeitung des ersten Satzes zur Verfügung stehen. Die Dateien mit den TBox-Definitionen werden in der Datei `mtmenu.pl` durch das Kürzel `tbox` identifiziert (siehe Abschnitt 3.5.1 und Anhang B). Die Beschreibung der Syntax der TBox-Definitionen hier würde den Rahmen dieser Arbeit sprengen. Deshalb verweise ich

auf das Handbuch für das BACK-System in [Hoppe et al. 93]. Es wird mit dem experimentellen MÜ-System per FTP oder Diskette kostenlos im POSTSCRIPT-Format zur Verfügung gestellt.

3 Benutzungshandbuch

Nach dem Laden des MÜ-Systems sind die Prädikate definiert, welche die Interaktion mit dem MÜ-System erlauben. Die in diesem Abschnitt beschriebenen Befehle können in der angegebenen Form nur im MÜ-System-Interpreter eingegeben werden. Allerdings ist ihre Eingabe ebenfalls im Prolog-Interpreter möglich (siehe dazu Abschnitt 3.2). Die Befehle können in folgende Klassen eingeteilt werden:

- Befehle für den MÜ-System-Interpreter,
- Aufruf der einzelnen Übersetzungsschritte,
- Aktionen mit gespeicherten Repräsentationen,
- Laden der linguistischen Daten und
- Veränderung von MÜ-System-Parametern.

Sämtliche Strukturabbildungen, bis auf die morphologische und syntaktische Analyse und die morphologische Synthese sind als Termersetzungssysteme realisiert. Die morphologische Analyse und Synthese basieren auf dem SUTRA-System (vgl. [Busemann 83a] und siehe Abschnitt 1.2.1). Der Parser (vgl. [Weisweber 87a], [Weisweber 87b], [Weisweber 88], [Weisweber/Preuß 92] und siehe Abschnitt 1.2.2) wurde auf der Grundlage einer konstruktiven Version der GPSG implementiert (vgl. [Busemann/Hauenschild 88], [Hauenschild/Busemann 88], [Preuß 89] und siehe Abschnitt 1.1.2). Die Grammatiken, die er verwendet, sind in [Preuß 87] und [Preuß 89] beschrieben.

Das MÜ-System ist mit sämtlichen linguistischen Daten für die Übersetzung eines Fragments Deutsch → Englisch initialisiert. Das Fragment und die vom System erzeugten Übersetzungen sind in Anhang A enthalten.

Im folgenden werden Variablen, die nur mit bestimmten im Text spezifizierten Werten instantiiert werden dürfen, *kursiv* geschrieben. Taste bedeutet, daß nacheinander die Tasten Taste zu drücken sind. Wird Taste angegeben, so ist nur die Taste mit der entsprechenden Bezeichnung auf der Tastatur zu drücken (z.B. Return). Für die Ausgaben des MÜ-Systems, Prolog-Kode und Dateinamen wird die Schriftart *courier* verwendet.

3.1 Starten des MÜ-Systems

Bevor das MÜ-System gestartet werden kann, muß der Quell-Kode vorher in das Verzeichnis `~/FAST-v94` kopiert worden sein (siehe Abschnitt 2.1). Es sollte nur aus dem Verzeichnis `~/FAST-v94/prog/mt` gestartet werden. Dazu ist es erforderlich, vor dem Start mit dem Befehl `cd ~/FAST-v94/prog/mt` in dieses Verzeichnis zu wechseln. Danach wird der Prolog-Interpreter aufgerufen, etwa mit dem Befehl prolog Return.⁸ Nach dem Start fordert der Interpreter durch den Prompt `'?-'` zur Eingabe auf und das MÜ-System kann durch die Eingabe [loadmt]. Return geladen und gestartet werden. Die Laderoutine in der Datei `loadmt.pl`⁹ stellt fest, welcher Interpreter vorliegt

⁸Das MÜ-System läuft mit Quintus- und SWI-Prolog. Quintus-Prolog ist ein kommerzielles Produkt und kann nur käuflich erworben werden. SWI-Prolog ist Public Domain Software und ist über einen FTP-Server erhältlich (siehe Abschnitt 2.1).

⁹Dateien ohne Verzeichnisangaben befinden sich im Verzeichnis `~/FAST-v94/prog/mt`.

und lädt die entsprechende Datei `loadmtQUI.pl` oder `loadmtSWI.pl`. In beiden Dateien ist festgelegt, in welcher Reihenfolge die Quell-Dateien des MÜ-Systems geladen werden. Diese Dateien können ersatzweise auch durch `[loadmtQUI], main.` `Return` bzw. `[loadmtSWI], main.` `Return` im Quintus- bzw. SWI-Prolog-Interpreter geladen werden. Der Quell-Kode wird in der folgenden Reihenfolge geladen:

1. das BACK-System (ein Wissensrepräsentationssystem zur Repräsentation des Text- und Hintergrundwissens als assertionales bzw. terminologisches Wissen, siehe [Hoppe et al. 93] und [Quantz/Kindermann 90]),
2. Prolog-Systemprädikate,
3. das MÜ-System und
4. die Anapherninterpretationskomponente (siehe Abschnitt 1.2.4).

Durch die Eingabe des Prädikats `main/0` wird der MÜ-System-Interpreter gestartet und es erfolgt zunächst folgende Meldung:

Welcome to the experimental translation system of the project KIT-FAST.

October 1993 by KIT-FAST Technical University of Berlin

write to: Sekr. FR 6-10
Franklinstr. 28/29
D-10587 Berlin
Germany

Telephone +49-30-314-73558 or (-73540) Fax +49-30-314-73622

E-Mail: `ww@cs.tu-berlin.de`

Anschließend wird abgefragt, welche Übersetzungsrichtung gewünscht wird. Für das MÜ-System ist nur die Richtung Deutsch → Englisch (Eingabe `g-e` `Return`) sinnvoll. Danach wird gefragt, ob die Dateien mit den linguistischen Defaultdaten, die in der Datei `mtmenu.pl` festgelegt sind (siehe Abschnitt 3.5.1 und Anhang B), geladen werden sollen oder nicht. Hier ist zunächst zu empfehlen, diese Frage mit ja (Eingabe `yes` `Return`) zu beantworten, da sonst alle Dateien einzeln eingegeben werden müssen. Anschließend werden die Dateien mit den linguistischen Daten geladen. Dabei wird genauso vorgegangen wie nach dem Befehl `data/0` (siehe Abschnitt 3.5.2 auf Seite 99).

Nach dem Laden der linguistischen Daten wird der Prompt `MT(AS --> ZS)>` des MÜ-Systems ausgegeben. `AS` und `ZS` spezifizieren die Ausgangs- bzw. Zielsprache. Der MÜ-System-Interpreter erlaubt eine komfortablere Eingabe als der Prolog-Interpreter.

3.2 Der MÜ-System-Interpreter

Der MÜ-System-Interpreter dient der komfortablen Benutzung des MÜ-Systems und zum Teil auch des Prolog-Interpreters. Seine besonderen Eigenschaften sind:

- Die Eingabe kann *ohne* den obligatorischen Punkt am Ende eines Terms erfolgen.

- Eine vereinfachte Notation *prädikat argument₀ ... argument_n* ist möglich. Die Argumente werden durch Leerzeichen voneinander getrennt. Beispielsweise müssen Dateinamen mit Pfadspezifikationen nicht in Anführungszeichen stehen. Als Argumente dürfen sämtliche Prolog-Terme außer die vereinfachte Notation selbst vorkommen. In einem Argument dürfen keine Leerzeichen vorkommen.
- Die Ausgabe nach Anfragen ist modifiziert.
- Für Prädikatnamen können Synonyme (Abkürzungen) definiert und verwendet werden (siehe Abschnitt 3.2.2).
- (System-)Prädikate können vom Backtracking ausgeschlossen werden (siehe Abschnitt 3.2.3).
- Eine erweiterbare Online-Dokumentation der MÜ-System-Befehle in Englisch steht zur Verfügung.
- Befehle im MÜ-System-Interpreter werden automatisch gespeichert und können einfach abgefragt und wiederholt werden.

Eine Referenzkarte mit den MÜ-System-Befehlen kann mithilfe des Befehls `mt_help/0` angezeigt werden (siehe Seite 70). Die Referenzkarte enthält kurze Informationen über alle möglichen MÜ-System-Befehle.

Sämtliche MÜ-System-Prädikate können sowohl in der üblichen Prolog-Notation *prädikat(argument₀, ..., argument_n)* als auch in der Form *prädikat argument₀ ... argument_n* eingegeben werden. Der in Prolog obligatorische Punkt am Ende jeder Eingabe darf fehlen. Ein Argument darf keine Leerzeichen enthalten. Es wird als Atom aufgefaßt, falls es keine andere Prolog-Datenstruktur darstellt. Dateinamen, die als Argumente vorkommen, müssen nicht in Hochkommata gesetzt werden.

Sämtliche Anfragen, die im MÜ-System-Interpreter eingegeben werden, werden in einem Speicher abgelegt und können auf einfache Weise wieder abgefragt und aufgerufen werden, falls der Speicher zwischenzeitlich nicht gelöscht wird (siehe Befehl `history/0`, `history_call/1` bzw. `history_delete/0` auf Seite 71, 71 bzw. 71).

Prinzipiell können im MÜ-System-Interpreter und im Prolog-Interpreter dieselben Anfragen eingegeben werden. Eine Ausnahme bilden Prädikate, die in der Datei `synonym.pl` als Synonyme (Abkürzungen) für Systemprädikate definiert werden (siehe Abschnitt 3.2.2 und Anhang C). Synonyme können als Prädikate in Anfragen nur im MÜ-System-Interpreter eingegeben werden.

Wenn die Erzeugung von Alternativen durch eine entsprechende Definition in der Datei `noback.pl` verhindert wurde (siehe Abschnitt 3.2.3 und Anhang D), erfolgt nur eine Meldung, ob die Anfrage erfüllt werden konnte oder nicht.

Ist eine Anfrage erfüllbar und wurde die Erzeugung von Alternativen nicht durch eine entsprechende Definition in der Datei `noback.pl` verhindert, so wird ausgegeben, wie oft sie erfüllbar ist. Kommen in der Anfrage Variablen vor, so werden danach die Variablenbelegungen ausgegeben. Kommen keine Variablen vor, so wird nur die Anfrage wiederholt. Nach der Ausgabe jeder Lösung wird gefragt, ob weitere Lösungen ausgegeben werden sollen. Falls eingegeben wird und keine weitere Lösung vorhanden ist, erfolgt eine entsprechende Meldung. Wird eingegeben, so werden keine weiteren Lösungen angezeigt.

Abschließend werden in diesem Abschnitt ein Befehl zur Erzeugung von komplexen Kategorien der verschiedenen Repräsentationsebenen (siehe Abschnitt 2.4.1), das Ausgabeformat von Repräsentationen, sowie die im MÜ-System verwendeten Datenwelten und Ergebniszähler vorgestellt.

3.2.1 MÜ-System-Befehle

`interpret.` `Return` oder `exit.` `Return`

Synonyme: keine definiert

Mit den Prädikaten `interpret/0` oder `exit/0` wird der MÜ-System-Interpreter aus dem Prolog-Interpreter gestartet. Prinzipiell kann jeder Befehl, der im folgenden beschrieben ist, auch direkt im Prolog-Interpreter verwendet werden. In diesem Fall muß die Anfrage jedoch in der üblichen Prädikat-Argument-Notation als Prolog-Term eingegeben werden und mit `.` `Return` abgeschlossen werden.

`mt_help` `Return`

Synonyme: `help/0`, `h/0`, `?/0`

Mit dem Prädikat `mt_help/0` werden kurze Informationen über die möglichen Befehle für das MÜ-System ausgegeben (Referenzkarte).

Die Referenzkarte wird zeilenweise angezeigt. Wird nach der Ausgabe einer Zeile nur die `Return`-Taste gedrückt, so wird die nächste Zeile ausgegeben. Erfolgt die Eingabe `q` `Return`, so wird die Ausgabe abgebrochen und zum MÜ-System-Interpreter zurückgekehrt.

`help_file +Befehl` `Return`

Synonyme: `help/1`, `h/1`, `?/1`

1. *Befehl* ist ein Atom (Name eines Prädikats oder eines definierten Synonyms bzw. ein Stichwort aus einer Datei für die Online-Dokumentation in Englisch).

Mit dem Prädikat `help_file/1` wird zunächst eine Liste der definierten Synonyme ausgegeben, falls das Argument ein Prädikatsname oder ein Synonym ist und Synonyme dafür definiert sind. Anschließend werden Informationen über den spezifizierten Befehlsnamen ausgegeben. Um die entsprechende Datei der Online-Dokumentation zu finden, wird zunächst in dem Verzeichnis, das mithilfe des Prädikats `help_directory/1` (siehe Seite 97) in der Datei `mtmenu.pl` festgelegt wurde (siehe Abschnitt 3.5.1 und Anhang B), eine Datei mit dem Namen *Befehl.help* gesucht. Ist sie vorhanden, so wird der Inhalt dieser Datei ausgegeben. Existiert diese Datei nicht, so werden sämtliche in diesem Verzeichnis vorhandenen Dateien nach dem Stichwort *Befehlsname* durchsucht. Anschließend wird eine Liste von `help_file`-Befehlen ausgegeben, die möglicherweise nähere Informationen über das angegebene Stichwort geben könnten.

Es können auch eigene Dateien *meinname.help* mit individuellen Informationen in dem durch das Prädikat `help_directory/1` festgelegten Verzeichnis definiert werden. Der Inhalt dieser Datei kann dann mit dem Befehl `help_file meinname.help` `Return` angezeigt werden.

Der Inhalt einer Hilfsdatei wird zeilenweise angezeigt. Wird nach der Ausgabe einer Zeile nur die `Return`-Taste gedrückt, so wird die nächste Zeile ausgegeben. Erfolgt die Eingabe `q` `Return`, so wird die Ausgabe abgebrochen und zum MÜ-System-Interpreter zurückgekehrt.

`load_data +Datei` `Return`

Synonyme: `ld/1`

1. *Datei* ist ein Atom (Name einer existierenden Datei).

Mit dem Prädikat `load_data/1` werden sämtliche Klauseln aus der angegebenen Datei dynamisch geladen, d.h. die geladenen Klauseln können aufgerufen und manipuliert werden. Der Dateiname kann ohne Anführungszeichen angegeben werden, auch wenn er Sonderzeichen wie '~', '.', '...' oder '/' enthält. Falls der Befehl im Prolog-Interpreter verwendet wird, muß ein Dateiname in Anführungszeichen stehen, wenn in ihm Sonderzeichen vorkommen.

`history` `Return`

Synonyme: `hist/0`

Mit dem Prädikat `history/0` wird der Inhalt des Befehlsspeichers ausgegeben. Sämtliche Befehle, die im MÜ-System-Interpreter eingegeben werden, werden durchnummeriert und gespeichert. Die Ausgabe der Befehle erfolgt mit den zugeordneten Nummern, welche die Wiederholung des entsprechenden Befehls mit `history_call Nummer` `Return` ermöglicht.

`history_call +Nummer` `Return`

Synonyme: `hc/1`

1. *Nummer* ist eine positive ganze Zahl (Nummer eines Befehls im Befehlsspeicher).

Mit dem Prädikat `history_call/1` können Befehle, die im Befehlsspeicher des MÜ-System-Interpreters stehen, wiederholt werden, indem die entsprechende Nummer ($Nummer \geq 1$) aus dem Speicher als Argument angegeben wird. Bei Bedarf kann die Nummer eines Befehls mit `history` `Return` ermittelt werden. Fehlermeldungen erfolgen, wenn das angegebene Argument eine negative ganze Zahl ist, wenn unter der angegebenen Nummer kein Befehl gespeichert ist oder wenn die Nummer des aktuellen Befehls als Argument angegeben wurde.

`history_delete` `Return`

Synonyme: `hd/0`

Mit dem Prädikat `history_delete/0` werden sämtliche im Speicher vorhandenen Befehle gelöscht und der Zähler für die Befehle wird wieder mit 1 initialisiert (siehe Zähler `history` im Abschnitt 3.2.7). Der unmittelbar auf diesen Befehl folgende Befehl erhält im Speicher somit die Nummer 1.

`shell` `Return`

Synonyme: keine definiert

Mit dem Prädikat `shell/0` kann direkt vom MÜ-System-Interpreter in das Betriebssystem gewechselt werden, ohne das MÜ-System im Arbeitsspeicher zu löschen. Im Betriebssystem können dann sämtliche dort möglichen Befehle verwendet werden. Anschließend kehrt man mit `exit` `Return` wieder zum MÜ-System-Interpreter zurück in den Zustand, in dem er verlassen wurde.

`quit` `Return`

Synonyme: `end/0`, `bye/0`

Mit den Prädikaten `end/0`, `quit/0` oder `bye/0` kann vom MÜ-System-Interpreter zum Prolog-Interpreter gewechselt werden, so daß dieser in üblicher Weise benutzt werden kann. Anschließend kehrt man mit `exit.` `Return` oder `interpret.` `Return` wieder zum MÜ-System-Interpreter zurück in den Zustand, in dem er verlassen wurde.

`halt` `Return`

Synonyme: keine definiert

Mit dem Prädikat `halt/0` wird das MÜ-System verlassen, d.h. sowohl der MÜ-System-Interpreter als auch der Prolog-Interpreter, und kehrt in das Betriebssystem zurück. Sämtliche in der Datenbasis gespeicherten Informationen gehen dabei verloren. Dies gilt insbesondere für die während der Sitzung vom MÜ-System erzeugten Strukturen. Es empfiehlt sich daher, vor dem Verlassen des MÜ-Systems zu überprüfen, ob noch Strukturen mit dem Befehl `file/1` (siehe Abschnitt 3.4.3 auf Seite 96) gesichert werden sollen. Zur Sicherheit wird vor dem Verlassen noch einmal überprüft, ob tatsächlich in das Betriebssystem zurückgekehrt werden soll. Erst wenn diese Frage mit `yes` `Return` beantwortet wird, wird das MÜ-System verlassen. Nach allen anderen Eingaben kehrt man wieder zurück in den MÜ-System-Interpreter.

3.2.2 Definition von Synonymen für Prädikate

Für jedes Prädikat, sowohl des MÜ- als auch des Prolog-Systems, können in der Datei `synonyms.pl` Synonyme definiert werden. In der Regel sind Synonyme Abkürzungen für Prädikatnamen. Es besteht aber auch die Möglichkeit, einen längeren Namen für ein Prädikat zu definieren. Für einige Prädikate sind in der Datei `synonyms.pl` bereits Synonyme vordefiniert (siehe Anhang C). Bei der Beschreibung der MÜ-System-Prädikate werden die vordefinierten Synonyme angegeben.

Ein Synonym für ein Prädikat wird mithilfe des Prädikats `synonym/3` definiert. Die definierten Synonyme sind nur im MÜ-System-Interpreter gültig:

`synonym(?Synonym, +Prädikat, +Stelligkeiten)`

1. *Synonym* ist ein Atom (Synonym für die spezifizierten Prädikate).

2. *Prädikat* ist ein Atom (Name eines im MÜ- oder Prolog-System vordefinierten Prädikats, für welches das Synonym definiert werden soll). Das Prädikat kann auch ein bereits definiertes Synonym sein.
3. *Stelligkeiten* ist eine Prolog-Liste mit positiven ganzen Zahlen (Stelligkeiten des spezifizierten Prädikats, für welches das Synonym definiert werden soll). Ist dieses Argument eine Variable, so ist das Synonym für das angegebene Prädikat mit jeder Stelligkeit definiert.

3.2.3 Ausschluß von Prädikaten vom Backtracking

Für jedes Prädikat, sowohl des MÜ- als auch des Prolog-Systems, kann festgelegt werden, daß es vom Backtracking ausgeschlossen werden soll. Dies erfolgt in der Datei `noback.pl`, in der die meisten Systemprädikate bereits eingetragen sind (siehe Anhang D). Wenn im MÜ-System-Interpreter eine Anfrage mit einem in dieser Datei spezifizierten Prädikat gestellt wird, so wird es in jedem Fall höchstens einmal erfüllt. Evtl. mögliche Variablenbelegungen werden bei Erfüllbarkeit nicht ausgegeben. Das gleiche gilt für alternative Ergebnisse. Dies ist sinnvoll für Prädikate wie `reconsult/1`, `debug/0`, `spy/1`, usw.

Ein MÜ-System- oder Prolog-Prädikat wird mithilfe des Prädikats `no_backtracking/3` vom Backtracking ausgeschlossen. Die entsprechenden Definitionen sind nur im MÜ-System-Interpreter gültig. Sie haben insbesondere keine Auswirkungen auf die Verwendung solcher Prädikate in Prolog-Programmen.

`no_backtracking(-System, +Prädikat, +Stelligkeiten)`

1. *System* ist ein Atom `mt` oder `prolog`, das festlegt, ob die spezifizierten Prädikate vordefinierte MÜ- oder Prolog-System-Prädikate sind.
2. *Prädikat* ist ein Atom (Name eines im MÜ- oder Prolog-System vordefinierten Prädikates, für welches nach der Eingabe im MÜ-System-Interpreter kein Backtracking durchgeführt werden soll).
3. *Stelligkeiten* ist eine Prolog-Liste mit positiven ganzen Zahlen (Stelligkeiten des spezifizierten Prädikats, für welche nach der Eingabe im MÜ-System-Interpreter kein Backtracking durchgeführt werden soll). Ist dieses Argument eine Variable, so wird für das angegebene Prädikat mit jeder Stelligkeit kein Backtracking durchgeführt.

3.2.4 Erzeugung komplexer Kategorien

<code>generate_category +Ebene ?Hauptkategorie +Spezifikationen ?Kategorie</code>	Return
---	---------------

Synonyme: `gc/4`, `gencat/4`

1. *Ebene* ist ein Atom (Repräsentationsebene):
 - `gpsg`: ausgangssprachliche GPSG-Kategorie,
 - `syms`: ausgangssprachlicher GPSG-Kategorienterm,
 - `hybs`: ausgangssprachlicher hybrider Kategorienterm,
 - `fass`: ausgangssprachlicher FAS-Kategorienterm,
 - `atls`: ausgangssprachlicher ATL-Kategorienterm,

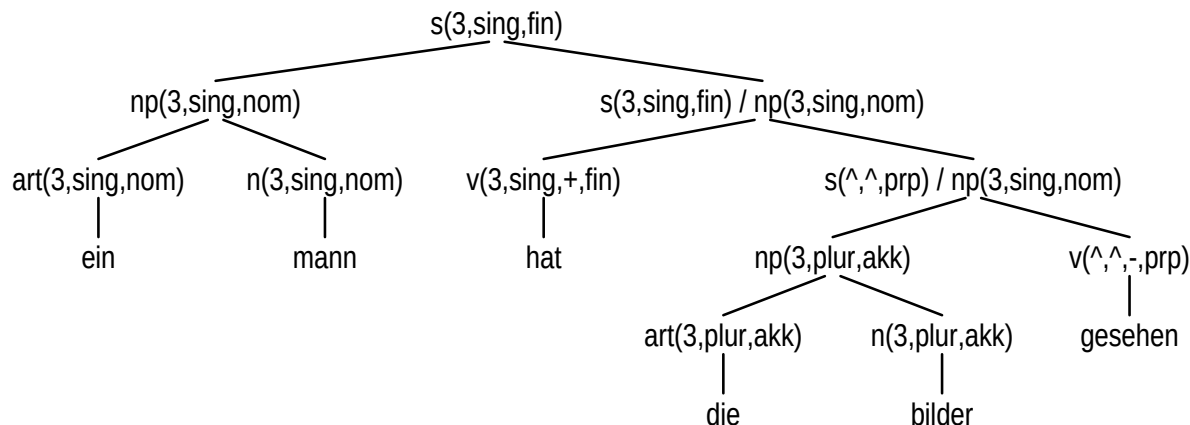


Abbildung 15: Beispiel für einen Ableitungsbaum in der üblichen Notation

- **fast**: zielsprachlicher FAS-Kategorienterm,
- **hybt**: zielsprachlicher hybrider Kategorienterm und
- **synt**: zielsprachlicher syntaktischer Kategorienterm.

2. *Hauptkategorie* ist ein Atom.
3. *Spezifikationen* ist eine Liste mit Merkmalspezifikationen der Form *+Merkmal* : *+Wert* für die Hauptkategorie mit
 - (a) *Merkmal* ist ein Atom.
 - (b) *Wert* ist ein Atom oder eine ganze Zahl (Wert des Merkmals).
4. *Kategorie* ist ein Term (komplexe Kategorie).

Mit dem Prädikat `generate_category/4` wird eine komplexe Kategorie oder ein komplexer Kategorienterm der angegebenen Ebene erzeugt. Die Hauptkategorie und eine Liste von Merkmalspezifikationen müssen ebenfalls als Argumente benannt werden. Mit einer Anfrage `generate_category Ebene H [] K Return` mit $Ebene \in \{\text{gpsg, syng, hybs, fass, atls, fast, hybt, synt}\}$ lassen sich sämtliche komplexe Kategorien der spezifizierten Ebene (siehe Abschnitt 2.4.1) generieren.

3.2.5 Form der Ausgabe von Ableitungsbäumen

Sämtliche im MÜ-System verwendeten linguistischen Repräsentationen sind Ableitungsbäume (Phrasenstrukturen) mit komplexen Kategorien als Knotenmarkierungen (siehe Abschnitt 1.1 und Abschnitt 2.4.1). Da die komplexen Kategorien insbesondere von GPSG-Strukturen in der Regel sehr lang sind, wäre eine Ausgabe der Ableitungsbäume in der üblichen Form (siehe Abbildung 15) sehr unübersichtlich.

Die in Abbildung 15 verwendeten komplexen Kategorien sind Terme erster Ordnung (Prolog-Terme), d.h. sie bestehen aus einem Prädikat und einer festgelegten Anzahl von Argumenten. Das Prädikat repräsentiert eine Hauptkategorie (**s** für Satz, **v** für Verb, **np** für Nominalphrase, **n** für Nomen und **art** für Artikel). Die Positionen der Argumente des Prädikats repräsentieren die Merkmale, die der entsprechenden Hauptkategorie zugeordnet sind, und die Argumente selbst repräsentieren ihre Werte.

```

s(3,sing,fin)
--- np(3,sing,nom)
----- art(3,sing,nom) ein
----- n(3,sing,nom) mann
--- s(3,sing,fin) / np(3,sing,nom)
----- v(3,sing,+,fin) hat
----- s(^,^,prp) / np(3,sing,nom)
----- np(3,plur,akk)
----- art(3,plur,akk) die
----- n(3,plur,akk) bilder
----- v(^,^,-,prp) gesehen

ein mann hat die bilder gesehen

```

Abbildung 16: Beispiel für einen Ableitungsbaum in indentierter Notation

Der Hauptkategorie **s** sind die Merkmale **person**, **numerus**, **verbform** und **slash** in der angegebenen Reihenfolge zugeordnet. Dabei wird der komplexe Wert des Merkmals **slash** nicht in der Form **s(3,sing,fin, np(3,sing,nom))** angegeben, sondern als **s(3,sing,fin) / np(3,sing,nom)**. Wenn der Wert des Merkmals **slash** undefiniert ('^') ist, wird die Spezifikation '/' weggelassen.

Den anderen Hauptkategorien sind folgende Merkmale in der angegebenen Reihenfolge zugeordnet:

- **s** → **person**, **numerus**, **verbform** und **slash**
- **v** → **person**, **numerus**, **auxiliar** und **verbform**
- **np** → **person**, **numerus** und **kasus**
- **n** → **person**, **numerus** und **kasus**
- **art** → **person**, **numerus** und **kasus**

Die im MÜ-System verwendeten Strukturen sind spezielle Ausprägungen von Ableitungsbäumen. Dies ist in der Regel immer dann der Fall, wenn ein Fragment einer natürlichen Sprache durch eine kontextfreie Grammatik beschrieben bzw. erzeugt wird. In Ableitungsbäumen, die durch solche Grammatiken erzeugt werden, dominiert jede Kategorie, die ein terminales Symbol (Wortform) dominiert, genau dieses eine Symbol, da jeder (lexikalischen) Kategorie in einem Ableitungsbaum genau eine Wortform zugeordnet ist. Diese Symbole werden auch oft als terminales und prä-terminales Symbol (oder Kategorie) bezeichnet.

Abbildung 16 zeigt den der Abbildung 15 entsprechenden Ableitungsbaum in indentierter Notation, die auch zur Ausgabe im MÜ-System verwendet wird. Dabei werden sämtliche Töchter einer Kategorie mit derselben Indentierung unter ihr ausgegeben. Die Indentierung der Tochterbäume ist um drei Bindestriche tiefer als die der Mutter. Die Ausgabe von oben nach unten entspricht der Reihenfolge von links nach rechts im Baum. Die Wortformen (Terminale) und ihre entsprechenden lexikalischen Kategorien werden in einer Zeile hintereinander geschrieben.

Die Ausgabe der Ableitungsbäume läßt sich durch drei MÜ-System-Parameter beeinflussen:

- Der Parameter **display** (siehe Seite 106) legt fest, ob die Strukturen während der syntaktischen Analyse oder der Termersetzung ausgegeben werden sollen oder nicht.
- Der Parameter **extend** (siehe Seite 106) steuert, ob als Argumente der Hauptkategorien nur die Werte der zugeordneten Merkmale oder die entsprechenden Merkmal-Wert-Paare ausgegeben werden.
- Der Parameter **alias** (siehe Seite 108) gibt an, ob die komplexen GPSG-Kategorien in Kurzform ausgegeben werden sollen oder nicht.

3.2.6 Datenwelten

Von den fünf Übersetzungsschritten, die im MÜ-System implementiert sind, werden vier durch dasselbe Verfahren realisiert (siehe Abschnitt 1 über die Architektur des MÜ-Systems und Abschnitt 1.2.3 über das Verfahren auf der Basis von Termersetzung). Ein solches System zu implementieren, erfordert besondere Rücksichtnahme auf die Datenverwaltung, denn für jeden Übersetzungsschritt existieren verschiedene Daten, die das gleiche Format haben. Deshalb werden die Daten separat in verschiedene Welten für die syntaktische, semantische und konzeptuelle Analyse, den Transfer und die Generierung geladen. Jede dieser Welten enthält sämtliche Daten, die der Interpretierer für den entsprechenden Übersetzungsschritt benötigt (siehe Abschnitt 3.5). Der Interpretierer bekommt einen zusätzlichen Parameter, der die Welt spezifiziert, deren Daten er im aktuellen Abbildungsschritt verwenden soll. Im MÜ-System existieren weitere Welten für die morphologische Analyse und Synthese, die Anapherninterpretationskomponente und für die Dominanzrelation, die vom Parser (siehe Abschnitt 1.2.2) verwendet wird. Die folgenden (Daten)Welten existieren im MÜ-System:

- **morpar** für die morphologische Analyse,
- **parser** für die syntaktische Analyse,
- **dom** für die Dominanz-Relation des Parsers,
- **gpsgfas** für die semantische Analyse,
- **sfasatl** für die konzeptuelle Analyse,
- **eval** für die Anapherninterpretation,
- **transfer** für den Transfer,
- **generator** für die Generierung und
- **morgen** für die morphologische Synthese.

Für die Arbeit mit (Daten)Welten werden zusätzliche Prädikate zur Verfügung gestellt. Im MÜ-System werden die Welten tatsächlich nur als Datenwelten verwendet, sie können aber auch als Kodewelten verwendet werden. Das heißt, daß neben Prolog-Fakten auch Prolog-Klauseln geladen werden können.

Die folgenden Prädikate wurden mithilfe des Modulkonzepts in Prolog definiert. Ein Prädikat *prädikat_w(Term, Welt)* kann aus diesem Grund auch durch *Welt : prädikat(Term)* aus dem Prolog-Interpreter aufgerufen werden. Der MÜ-System-Interpreter (siehe Abschnitt 3.2) kann die letztere Anfrage jedoch nur verstehen, wenn vor und nach dem Doppelpunkt ein Leerzeichen vorkommt und *Term* ein echter Prolog-Term ist. Die Prädikate **reconsult_w/2** und **reconsult/1** unterscheiden sich jedoch, weil das erste Klauseln und Fakten dynamisch in die Datenbasis lädt und das zweite statisch.

reconsult_w <i>+Datei +Welt</i>	Return
--	---------------

Synonyme: keine definiert

1. *Datei* ist ein Atom (Name einer Datei).
2. *Welt* ist ein Atom (Name einer Welt).

Das Prädikat **reconsult_w/2** lädt die Klauseln und Fakten aus der angegebenen Datei dynamisch in die spezifizierte Welt der Datenbasis und löscht dort bereits vorhandene Klauseln und Fakten mit derselben Stelligkeit, falls erforderlich.

consult_w <i>+Datei +Welt</i>	Return
--------------------------------------	---------------

Synonyme: keine definiert

1. *Datei* ist ein Atom (Name einer Datei).
2. *Welt* ist ein Atom (Name einer Welt).

Das Prädikat **consult_w/2** lädt die Klauseln und Fakten aus der angegebenen Datei dynamisch in die spezifizierte Welt der Datenbasis und fügt sie an das Ende aller in dieser Welt bereits vorhandenen Daten an.

call_w <i>+Anfrage +Welt</i>	Return
-------------------------------------	---------------

Synonyme: keine definiert

1. *Anfrage* ist ein Term (Anfrage).
2. *Welt* ist ein Atom (Name einer Welt).

Das Prädikat **call_w/2** versucht, die Anfrage in der spezifizierten Welt zu erfüllen.

listing_w <i>+Prädikat +Welt</i>	Return
---	---------------

Synonyme: keine definiert

1. *Prädikat* ist ein Atom (Name eines Prädikats) oder ein Term der Form *+Prädikat / +Stelligkeit*. Dieses Argument ist optional.
2. *Welt* ist ein Atom (Name einer Welt).

Das Prädikat `listing_w/2` gibt sämtliche Klauseln und Fakten, deren Kopf ein Term mit dem angegebenen Prädikatsnamen ist und die in der spezifizierten Welt der Datenbasis gespeichert sind, auf dem Bildschirm aus. Falls die Stelligkeit angegeben wird, werden nur die Terme mit der entsprechenden Stelligkeit berücksichtigt.

Das Prädikat `listing_w/1` gibt sämtliche Klauseln und Fakten, die in der spezifizierten Welt der Datenbasis gespeichert sind, auf dem Bildschirm aus.

<code>assert_w +Klausel +Welt</code>	Return
--------------------------------------	--------

Synonyme: keine definiert

1. *Klausel* ist ein Term.
2. *Welt* ist ein Atom (Name einer Welt).

Das Prädikat `assert_w/2` fügt die angegebene Klausel (Regel oder Faktum) an unbestimmter Stelle in die spezifizierte Welt der Datenbasis ein.

Die Prädikate `asserta_w/2` und `assertz_w/2` fügen die Klausel an erster bzw. letzter Stelle in die Datenbasis ein.

<code>retract_w +Klauselschema +Welt</code>	Return
---	--------

Synonyme: keine definiert

1. *Klauselschema* ist ein Term (Schema für eine Klausel, d.h. Regel oder Faktum).
2. *Welt* ist ein Atom (Name einer Welt).

Das Prädikat `retract_w/2` entfernt die erste Klausel aus der spezifizierten Welt der Datenbasis, die mit dem angegebenen Schema unifizierbar ist. Dieses Prädikat kann mehrfach erfüllt werden, indem die nächstmögliche Klausel entfernt wird, solange bis sämtliche Klauseln entfernt sind.

Das Prädikat `retractall_w/2` entfernt sämtliche Klauseln mit dem angegebenen Schema aus der spezifizierten Welt der Datenbasis.

3.2.7 Zähler für Ergebnisse

Die Ergebnisse der syntaktischen, semantischen und konzeptuellen Analyse, des Transfers sowie der Generierung werden in der Datenbasis (Defaultwelt `user`) gespeichert. Dabei bekommt jede Ergebnisstruktur ein eindeutiges Label (siehe Abschnitt 3.3). Die Label werden mithilfe entsprechender Zähler vom MÜ-System generiert, die beim Start des Systems mit 0 initialisiert werden.

Außerdem existieren im MÜ-System zwei weitere Zähler. Der eine dient zur eindeutigen Nummerierung der Befehle im Befehlsspeicher (siehe Abschnitt 3.2). Der andere zählt die Anzahl der Alternativen bei Anfragen im MÜ-System-Interpreter (siehe Abschnitt 3.2). Insgesamt gibt es im MÜ-System Zähler mit folgenden Namen:

- `gpsg` für Ergebnisse der syntaktischen Analyse,
- `fas` für Ergebnisse der semantischen Analyse,
- `at1` für Ergebnisse der konzeptuellen Analyse,

- **gen** für Ergebnisse des Transfers,
- **res** für Ergebnisse des Generators,
- **history** zur Nummerierung der Befehle im Befehlsspeicher und
- **backtrack** zum Zählen der Alternativen bei Anfragen im MÜ-System-Interpreter

Vom Benutzer des MÜ-Systems können auch zusätzliche Zähler definiert werden. Sämtliche Zähler können mit den folgenden Befehlen abgefragt und manipuliert werden.

<code>ctr_set +Zähler +Wert</code>	Return
------------------------------------	---------------

Synonyme: keine definiert

1. *Zähler* ist ein Atom oder ein Term (Name eines Zählers).
2. *Wert* ist eine ganze Zahl (Wert des Zählers).

Das Prädikat `ctr_set/2` richtet den angegebenen Zähler ein, falls erforderlich, und setzt ihn auf den spezifizierten Wert.

<code>ctr_is +Zähler -Wert</code>	Return
-----------------------------------	---------------

Synonyme: keine definiert

1. *Zähler* ist ein Atom oder ein Term (Name eines Zählers).
2. *Wert* ist eine ganze Zahl (Wert des Zählers).

Das Prädikat `ctr_is/2` unifiziert den spezifizierten Wert oder die Variable mit dem aktuellen Wert des angegebenen Zählers, falls der Zähler vorhanden ist.

<code>ctr_inc +Zähler -Wert</code>	Return
------------------------------------	---------------

Synonyme: keine definiert

1. *Zähler* ist ein Atom oder ein Term (Name eines Zählers).
2. *Wert* ist eine ganze Zahl (Wert des Zählers).

Das Prädikat `ctr_inc/2` unifiziert den spezifizierten Wert oder die Variable mit dem aktuellen Wert des angegebenen Zählers, falls der Zähler vorhanden ist, und addiert danach den Wert 1.

<code>ctr_dec +Zähler -Wert</code>	Return
------------------------------------	---------------

Synonyme: keine definiert

1. *Zähler* ist ein Atom oder ein Term (Name eines Zählers).
2. *Wert* ist eine ganze Zahl (Wert des Zählers).

Das Prädikat `ctr_dec/2` unifiziert den spezifizierten Wert oder die Variable mit dem aktuellen Wert des angegebenen Zählers, falls der Zähler vorhanden ist, und subtrahiert anschließend den Wert 1.

3.3 Aufruf der Übersetzungsschritte

Sämtliche in diesem Abschnitt beschriebenen Übersetzungsschritte bis auf die morphologische Analyse und Synthese sowie die syntaktische Analyse sind durch ein Termersetzungsverfahren realisiert (siehe Abschnitt 1.2.3). Deshalb wird zunächst der Ablauf eines einzelnen Übersetzungsschrittes mit dem Termersetzungsverfahren schematisch dargestellt. Anschließend wird der Aufruf und Ablauf der morphologischen, syntaktischen, semantischen und konzeptuellen Analyse, des Transfers, der Generierung, der morphologischen Synthese und der Übersetzung eines Satzes durch Hintereinanderausführung der einzelnen Komponenten vorgestellt.

3.3.1 Strukturabbildung mit dem Termersetzungsverfahren

Ein Übersetzungsschritt mit dem Termersetzungsverfahren läuft immer nach demselben Schema ab. Nach dem Aufruf eines Schrittes wird die Eingabestruktur wiederholt und die Kette der terminalen Symbole der entsprechenden Struktur ausgegeben. Danach erfolgt der jeweilige Übersetzungsschritt. Wenn dieser zu keiner Zielstruktur führt, wird eine entsprechende Meldung ausgegeben. Andernfalls werden die Zielstrukturen ausgegeben. Für die Ausgabe werden die Ergebnisse beginnend mit 1 fortlaufend durchnummeriert.

Jedes Ergebnis wird in der Datenbasis mithilfe der Prädikate **fas**/2 (semantische Analyse), **at1**/2 (konzeptuelle Analyse), **gen**/2 (Transfer) und **res**/2 (Generierung) gespeichert. Das zweite Argument jedes dieser Prädikate ist die Ergebnisstruktur. Das erste Argument ist ein vom MÜ-System-Interpreter eindeutig vergebenes Label **fas n** , **at1 n** , **gen n** bzw. **res n** . Dabei ist n eine natürliche Zahl, die der Wert des Zählers mit dem Namen **fas**, **at1**, **gen** bzw. **res** ist. Nach der Ableitung eines Ergebnisses durch das Termersetzungsverfahren in einem Übersetzungsschritt wird der entsprechende Zähler automatisch inkrementiert. Bevor eine Ergebnisstruktur und die zugehörige Kette von terminalen Symbolen ausgegeben werden, wird angezeigt

- die Nummer des Ergebnisses für diesen Übersetzungsschritt sowie
- das Prädikat und das Label, mit denen es in der Datenbasis gespeichert wurde.

Die Ergebniszähler können abgefragt und manipuliert werden (siehe Abschnitt 3.2.7). Auf die gleiche Art werden auch die Ergebnisse der syntaktischen Analyse in der Datenbasis gespeichert (siehe Abschnitt 3.3.3).

Die vom Parser und vom Termersetzungsmodul gespeicherten Ergebnisse dienen als Eingabe für die jeweils nachfolgenden Übersetzungsschritte. Außerdem können sie angezeigt oder gelöscht bzw. in einer Datei gespeichert werden (siehe Abschnitt 3.4).

Nach der Ausgabe aller Ergebnisstrukturen wird die Anzahl der Ergebnisse des jeweiligen Übersetzungsschrittes für eine Eingabestruktur ausgegeben. Anschließend kann die Ableitung eines der Ergebnisse in Einzelschritten nachvollzogen werden. Um diese Option zu wählen, ist die entsprechende Systemanfrage positiv zu beantworten (☐ **yes** ☐ **Return**). Andernfalls kehrt das System auf die Interpretierebene zurück.

Wurde diese Option gewählt, so muß weiterhin spezifiziert werden, welches der Ergebnisse genauer betrachtet werden soll. Dazu ist seine Nummer einzugeben. Danach kann die Abbildung der Eingabestruktur in die angegebene Zielstruktur in einzelnen Schritten nachvollzogen werden. Dies ist besonders dann sinnvoll, wenn für eine Eingabestruktur eine intendierte Zielstruktur mithilfe des jeweiligen Termersetzungssystems nicht abgeleitet werden kann. In diesem Fall kann die Überprüfung einer Ergebnisstruktur gemäß

der Zielsyntax nach der Anwendung sämtlicher anwendbaren Abbildungsregeln ausgeschaltet werden (siehe System-Parameter **syntax** auf Seite 114), so daß danach auch Ergebnisstrukturen ausgegeben werden, die nicht der Zielsyntax entsprechen. Die Abbildungsschritte bis zur Erzeugung dieser Ergebnisstrukturen können dann nachvollzogen werden.

Nachdem das Termersetzungsverfahren in dem Modus für die Anzeige einzelner Ableitungsschritte ist, wird nach der Anwendung jeder Termersetzungsregel deren Nummer und das Ergebnis der Anwendung ausgegeben. Dabei werden die Stellen, an denen die Regel angewendet wurde, mit '#' markiert.

Nach jedem einzelnen Abbildungsschritt besteht die Möglichkeit, einen Schritt zurück oder vorwärts zu gehen bzw. die Ausgabe der einzelnen Abbildungsschritte zu beenden. Dies erfolgt durch die Eingabe der Zeichen **p** (*previous*), **n** (*next*) bzw. **e** (*exit*).

3.3.2 Morphologische Analyse einer Wortform

<code>morphological_analysis + Wortform</code>	<code>Return</code>
--	---------------------

Synonyme: `ma/1`

1. *Wortform* ist ein Atom (Wortform der Ausgangssprache).

Mit dem Prädikat `morphological_analysis/1` wird die angegebene Wortform der Ausgangssprache morphologisch analysiert. Diese Komponente basiert auf dem SUTRA-System (siehe Abschnitt 1.2.1). Sämtliche Lesarten werden in Form der entsprechenden lexikalischen GPSG-Kategorie auf dem Bildschirm angezeigt. Die Form der Ausgabe kann durch die System-Parameter **alias** (siehe Seite 108) und **extend** (siehe Seite 106) verändert werden.

Beispiellauf für die morphologische Analyse einer Wortform

```
MT(g-->e)> morphological_analysis ergreift
v(6,2,+,^,-,^,ACO,+,fin,-) imp
v(6,3,-,^,-,^,ACO,MC0,fin,-) pres
v(6,2,+,^,-,^,ACO,MC0,fin,-) pres

>>> ok
```

3.3.3 Syntaktische Analyse eines Satzes

<code>parse</code>	<code>Return</code>
--------------------	---------------------

Synonyme: `p/0`, `dcp/0`, `parser/0`, `parsing/0`

Mit dem Prädikat `parse/0` wird ein einzugebender Satz der Ausgangssprache syntaktisch analysiert. Zunächst erfolgt dabei die Eingabe eines Satzes der Ausgangssprache, die mit `Return` beendet wird. Danach wird der Satz morphologisch analysiert. Die System-Parameter, die für den Parser gesetzt werden können, sind im Abschnitt 3.6.2 beschrieben. Der Parser (siehe Abschnitt 1.2.2) erzeugt sämtliche mögliche GPSG-Strukturen für einen eingegebenen Satz.

Während der morphologischen Analyse werden die Lesarten der Wortformen, die im Satz vorkommen, auf dem Bildschirm ausgegeben. Falls eine eingegebene Wortform nicht morphologisch analysiert werden kann, wird eine Korrekturmöglichkeit eingeleitet, in der

das Wort verbessert oder gelöscht werden kann. Nach der morphologischen Analyse wird der Parser gestartet. Wenn die syntaktische Analyse erfolgreich war, werden alle Lösungen in einer indentierten Baumnotation mit komplexen Kategorien ausgegeben (siehe Abschnitt 3.2.5). Nach der Ausgabe jedes Baumes wird angezeigt, unter welchem Prädikat (hier `gpsg/2`) und mit welchem Label (hier `gpsgn`) der Baum in der Datenbasis abgespeichert wurde. Das Label `gpsgn` wird vom MÜ-System für jedes Ergebnis eindeutig vergeben. Diese Ausgabe ist wichtig, um das Ergebnis weiterverarbeiten (siehe Befehl `semantic_analysis/{0,1}`) oder später noch einmal anzeigen bzw. löschen zu können (siehe Befehl `pretty_print/{1,2}` bzw. `delete/{1,2}`).

Falls die syntaktische Analyse nicht erfolgreich war, erfolgt eine entsprechende Meldung. In diesem Fall können die im Abschnitt 3.6.2 aufgeführten System-Parameter für den Parser gesetzt werden. Sie ermöglichen die Wiederholung der syntaktischen Analyse mit demselben Satz, um dabei mithilfe der zusätzlichen Ausgaben die Fehler zu lokalisieren.

Beispiellauf für die syntaktische Analyse eines Satzes

```
MT(g-->e)> parse
>>> Please input a sentence ending with "[RETURN]" !

Input> sie erhoehen ihre investitionen

>>> Lemmatization for "sie".
d(1,3,PLU0,GEN0,acc,FLEX0)
d(1,3,PLU0,GEN0,nom,FLEX0)
-----
>>> Lemmatization for "erhoehen".
v(6,PERO,PLU0,THEM0,-,^,-,inf,-) inf
v(6,1,+,^,-,ACO,MC0,fin,-) pres
v(6,3,+,^,-,ACO,MC0,fin,-) pres
v(6,1,+,^,-,ACO,MC0,fin,-) kon1
v(6,3,+,^,-,ACO,MC0,fin,-) kon1
-----
>>> Lemmatization for "ihre".
d(2,3,-,2,nom,1)
d(2,3,-,2,acc,1)
d(2,3,+,GEN0,nom,1)
d(2,3,+,GEN0,acc,1)
-----
>>> Lemmatization for "investitionen".
n(5,+,2,CAS0)
-----

>>> Input: sie erhoehen ihre investitionen

>>> dominance chart parser running ...

v3(3,+,THEM0,-,+,+,fin,-)
--- dp(3,PLU1,acc,top)
----- d1(3,PLU1,acc)
----- d(1,3,PLU1,GEN1,acc,FLEX1) sie
--- v3(3,+,^,-,+,+,fin,-)/dp(3,PLU1,acc,top)
----- v(6,3,+,^,-,^,+,+,fin,-) erhoeh
----- dp(3,+,nom,THEM7)
----- d1(3,+,nom)
```

```

----- d(2,3,+,2,nom,1) ihr
----- np(+,nom)
----- n(5,+,2,nom) investition

>>> Result stored with label "gpsg2" under predicate "gpsg/2" !

v3(3,+,THEM0,-,+,+,fin,-)
--- dp(3,+,nom,top)
----- d1(3,+,nom)
----- d(1,3,+,GEN1,nom,FLEX1) sie
--- v3(3,+,^,-,+,+,fin,-)/dp(3,+,nom,top)
----- v(6,3,+,^,-,^,+,+,fin,-) erhoeht
----- dp(3,+,acc,THEM7)
----- d1(3,+,acc)
----- d(2,3,+,2,acc,1) ihr
----- np(+,acc)
----- n(5,+,2,acc) investition

>>> Result stored with label "gpsg3" under predicate "gpsg/2" !

>>> Analysis terminates successfully !

>>> ok

```

3.3.4 Semantische Analyse eines Satzes

<code>semantic_analysis +Label</code>	<code>Return</code>
---------------------------------------	---------------------

Synonyme: $s/\{0,1\}$, $sa/\{0,1\}$

1. *Label* ist ein Atom `gpsgn` (Label einer gespeicherten GPSG-Struktur der Ausgangssprache). Das Argument ist optional und wird interaktiv abgefragt, falls es nicht angegeben wird.

Mit dem Prädikat `semantic_analysis/{0,1}` wird die GPSG-Struktur der Ausgangssprache (Ableitungsbaum mit GPSG-Kategorien) mit dem angegebenen Label semantisch analysiert. Zunächst wird dabei die GPSG-Struktur ausgegeben oder es kommt die Meldung, daß in der Datenbasis keine Struktur mit dem angegebenen Label gespeichert ist.

Die ausgegebene GPSG-Struktur sieht anders aus als die Ausgabe nach der syntaktischen Analyse (siehe Abschnitt 3.3.3). Der Grund liegt in der Einsparung eines Lexikonzugriffs, da das GPSG- und das FAS-Lexikon in Form eines Abbildungslexikons zusammengefaßt wurden. Auf dieses Lexikon wird für jede Wortform während der syntaktischen Analyse einmal zugegriffen. Nach erfolgreicher syntaktischer Analyse werden die GPSG-Lexikoneinträge als terminale und pre-terminale Symbole eines Ableitungsbau-
mes ausgegeben. Vor dem Abspeichern einer GPSG-Struktur in die Datenbasis werden die GPSG-Lexikoneinträge jedoch durch die entsprechenden FAS-Lexikoneinträge ersetzt und gleichzeitig sämtliche GPSG-Kategorien durch entsprechende GPSG-Terme ersetzt, um sie als Eingabe für das Termersetzungsverfahren zur Verfügung zu stellen. Dadurch werden Lexikonzugriffe in der semantischen Analyse eingespart, denn das Abbildungslexikon $\text{GPSG} \rightarrow \text{FAS}$ wäre normalerweise Bestandteil der semantischen Analyse.

Nach der Ausgabe der GPSG-Struktur mit dem angegebenen Label wird die Kette der terminalen Symbole der entsprechenden Struktur ausgegeben und es erfolgt die semantische Analyse durch das Termersetzungsverfahren (siehe Abschnitt 1.2.3), d.h. die

Abbildung einer ausgangssprachlichen GPSG-Struktur in einen oder mehrere äquivalente ausgangssprachliche FAS-Ausdrücke (Ableitungsbäume mit FAS-Kategorien). Nach deren erfolgreichen Abschluß werden die FAS-Ausdrücke in indentierter Baumnotation und die entsprechenden Ketten von terminalen Symbolen ausgegeben (siehe Abschnitt 3.2.5). Für die Ausgabe werden die Ergebnisse beginnend mit 1 fortlaufend durchnummeriert. Vor der Ausgabe wird die Nummer eines Ergebnisses angezeigt.

Nach der Ausgabe jedes FAS-Ausdrucks wird angezeigt, unter welchem Prädikat (hier `fas/2`) und mit welchem Label (hier `fasn`) der Ausdruck in der Datenbasis abgespeichert wurde. Das Label `fasn` wird vom MÜ-System für jedes Ergebnis eindeutig vergeben. Diese Ausgabe ist wichtig, um das Ergebnis weiterverarbeiten (siehe Befehl `conceptual_analysis/{0,1}`) oder später noch einmal anzeigen bzw. löschen zu können (siehe Befehl `pretty_print/{1,2}` bzw. `delete/{1,2}`).

Anschließend wird ausgegeben, wieviele Ergebnisse die semantische Analyse für eine Eingabestruktur erzeugen konnte und es besteht die Möglichkeit, die Ableitung eines der Ergebnisse in Einzelschritten nachzuvollziehen (siehe Abschnitt 3.3.1). Dies ist insbesondere dann wichtig, wenn die syntaktische Kontrolle nach der Termersetzung, die durch den System-Parameter `syntax` (siehe Seite 114) gesteuert wird, vorher abgeschaltet wurde, weil mit syntaktischer Kontrolle kein Ergebnis erzeugt werden konnte.

Beispiellauf für die semantische Analyse eines Satzes

```
MT(g-->e)> semantic_analysis gpsg3

v3(gpsg,THEM,-,+,+,fin,-,~,~)
--- dp(gpsg,nom,top,~,~)
----- d1(gpsg,~)
----- term(g,det,ROLE,THEM,3,plur,GENUS,object,REF,pron,NTYP,REAL,EXT,ACT) sie
--- v3(gpsg,~,~,+,+,fin,-,~,dp(gpsg,nom,top,~,~),~)
----- v_pred(hyb,lex,ag_af,active,-,THEM1,act,event,REF,6,~,~,+,+) erhoehen
----- dp(gpsg,acc,THEM,~,~)
----- d1(gpsg,~)
----- term(g,det,ROLE,THEM,3,NUM,GENUS,object,REF,pron,NTYP,REAL,EXT,ACT)
            ihr
----- np(gpsg,~)
----- n_pred(hyb,lex,ag_aim,3,plur,f,object,REF,noun,count,no,temp,no,5)
            investition

sie erhoehen ihr investition

>>> Rewrite result no 1
>>> Result stored with label "fas2" under predicate "fas/2" !

fas(g,illoc)
--- illoc(g,lex,ass) assertion
--- clausefin(g,fin,-,+,~,~,act,event,REF)
----- fin(g,lex,+,~,~) pres_ind
----- clause(g,pred,nil,~,nil,act,event,REF)
----- v_pred(g,lex,ag_af,active,-,2,act,event,REF) erhoehen
----- term(g,det,agent,1,3,plur,GENUS,object,REF,pron,NTYP,REAL,EXT,actor) sie
----- term(g,det,affected,3,3,plur,f,object,REF,noun,count,no,temp,no)
----- det(g,lex,3,plur,f,count,def,REF) *
----- nom(g,pred,3,plur,f,object,REF,noun,count,no,temp,no)
----- n_pred(g,lex,ag_aim,3,plur,f,object,REF,noun,count,no,temp,no)
            investition
----- term(g,det,agent,nil,3,NUM,GENUS,object,REF,pron,NTYP,REAL,EXT,
            ACT) ihr
```

```

assertion pres ind erhoeihen sie * investition ihr

>>> 1 rewrite result(s) !
>>> Do you want to trace one of the rewrite results ?
>>> Input: y(es) [RETURN] or n(o) [RETURN] > n

>>> ok

```

3.3.5 Konzeptuelle Analyse eines Satzes

<code>conceptual_analysis +Label</code>	Return
---	--------

Synonyme: $c/\{0,1\}$, $ca/\{0,1\}$

1. *Label* ist ein Atom *fasn* (Label eines gespeicherten FAS-Ausdrucks der Ausgangssprache). Das Argument ist optional und wird interaktiv abgefragt, falls es nicht angegeben wird.

Mit dem Prädikat `conceptual_analysis/{0,1}` wird der FAS-Ausdruck der Ausgangssprache (Ableitungsbaum mit FAS-Kategorien) mit dem angegebenen Label konzeptuell analysiert. Zunächst wird dabei der FAS-Ausdruck ausgegeben oder es kommt die Meldung, daß in der Datenbasis kein FAS-Ausdruck mit dem Label gespeichert ist.

Nach der Ausgabe des FAS-Ausdrucks wird die Kette der terminalen Symbole des entsprechenden Ausdrucks ausgegeben und es erfolgt die konzeptuelle Analyse durch das Termersetzungsverfahren (siehe Abschnitt 1.2.3), d.h. die Abbildung eines ausgangssprachlichen FAS-Ausdrucks in einen oder mehrere äquivalente ausgangssprachliche ATL-Bäume (Ableitungsbäume mit ATL-Kategorien).¹⁰ Nach dem erfolgreichen Abschluß werden die ausgangssprachlichen ATL-Bäume in indentierter Baumnotation und die entsprechenden Ketten von terminalen Symbolen ausgegeben (siehe Abschnitt 3.2.5). Für die Ausgabe werden die Ergebnisse beginnend mit 1 fortlaufend durchnummeriert. Vor der Ausgabe wird die Nummer eines Ergebnisses angezeigt.

Nach der Ausgabe jedes ATL-Baums wird angezeigt, unter welchem Prädikat (hier `atl/2`) und mit welchem Label (hier `atln`) der Ausdruck in der Datenbasis abgespeichert wurde. Das Label `atln` wird vom MÜ-System für jedes Ergebnis eindeutig vergeben. Diese Ausgabe ist wichtig, um das Ergebnis später noch einmal anzeigen bzw. löschen zu können (siehe Befehl `pretty_print/{1,2}` bzw. `delete/{1,2}`).

ATL-Bäume sind äquivalent zu ATL-Ausdrücken des BACK-Systems, aber sie entsprechen noch nicht der ATL-Syntax. Deshalb werden sie System-intern in einen äquivalenten ATL-Ausdruck abgebildet, welcher der Syntax des BACK-Systems entspricht. Dabei werden die Variablen `REF` eines FAS-Ausdrucks für die Referenzen der entsprechenden Diskursobjekte in der ABox übernommen. Anaphorische Ausdrücke (Personal- und Possessivpronomina) und sämtliche Informationen über sie werden zunächst nicht übernommen, sondern separat repräsentiert. Danach wird der ATL-Ausdruck an das BACK-System übergeben und die repräsentierte Information in die ABox eingetragen. Jedes Diskursobjekt wird als ABox-Objekt mit einer eindeutigen ABox-Referenz `uc(n)`, die vom BACK-System vergeben wird, repräsentiert. Der ATL-Ausdruck wird nach seinem Eintrag in die ABox mit den instantiierten ABox-Referenzen ausgegeben.

¹⁰Das im MÜ-System vordefinierte Termersetzungssystem für die konzeptuelle Analyse ist eindeutig. Dies ist adäquat für das im Anhang A enthaltene Fragment. Allgemein muß für die konzeptuelle Analyse jedoch auch von nicht eindeutigen Regelsystemen ausgegangen werden.

Danach gibt es zwei Möglichkeiten. Wenn der Satz keine Personal- oder Possessivpronomina enthält, wird der eingegebene FAS-Ausdruck mit den ABox-Referenzen aktualisiert. Kommen im Satz Pronomen vor, dann werden zunächst deren Antezedenten, d.h. die Diskursobjekte, auf die sie sich beziehen, ermittelt (vgl. [Dunker/Umbach 93]). Dazu wird jeder Ausdruck für ein Personal- oder Possessivpronomen und seine Bezüge zu möglichen Antezedenten einzeln ausgewertet. Das Auswertungsergebnis wird als Tabelle ausgegeben, die folgende Form hat:

<u>PRONOUNS</u>	<u>POSSIBLE ANTECEDENTS ...</u>
<i>Pronomen</i>	<i>Liste der Antezedenten</i>
var	<i>ABox-Referenz für das jeweilige Antezedens</i>
<i>Kriterium₁</i>	<i>Bewertung des jeweiligen Antezedenten nach Kriterium₁</i>
⋮	⋮
<i>Kriterium_n</i>	<i>Bewertung des jeweiligen Antezedenten nach Kriterium_n</i>
-----	-----
sum :	<i>Summe der Bewertungen des jeweiligen Antezedenten</i>
best antecedents :	<i>Antezedens₁ ABox-Referenz₁</i>
	⋮
	<i>Antezedens_m ABox-Referenz_m</i>

Der Eintrag **var** als ABox-Referenz bedeutet, daß für das Pronomen oder das jeweilige Antezedens noch keine ABox-Referenz ermittelt wurde. Da die Referenz für das linke obere Pronomen gerade ermittelt wird, bekommt es als ABox-Referenz den Eintrag **var**. Nach Abschluß der Auswertung können mehrere Antezedenten die beste (höchste) Bewertung haben, so daß eine Liste der besten Antezedenten ausgegeben wird. Das Pronomen bekommt dieselbe ABox-Referenz wie das erste in dieser Liste aufgeführte Antezedens.

Die im MÜ-System vordefinierten Kriterien für die Bewertung anaphorischer Bezüge sind zur Zeit **kongruenz**, **p_bindungen**, **naehe**, **subjekt**, **satzthema**, **rolle**, **freie_ergaenzung** und **diskurs** (näheres siehe [Hauenschild 91], [Schmitz et al. 92], [Preuß et al. 92] und [Preuß et al. 93]). Die Bewertungen sind (positive und negative) ganze Zahlen.

Für das Anapherninterpretationsverfahren können bestimmte Eingabe- und System-Parameter gesetzt werden (siehe [Dunker/Umbach 93] und Abschnitt 2.4.9 bzw. 3.6.4).

Nachdem für jedes Personal- und Possessivpronomen ein Antezedens ermittelt wurde, wird der eingegebene FAS-Ausdruck aktualisiert und die Informationen über die Pronomina werden unter dem ABox-Objekt eingetragen, auf den das entsprechende Antezedens mit seiner ABox-Referenz verweist. Dabei handelt es sich um eine Liste von ATL-Ausdrücken, welche die ABox-Referenz des ermittelten Antezedenten enthalten. Die Liste der in die ABox eingetragenen ATL-Ausdrücke wird ausgegeben.

Die Aktualisierung des eingegebenen FAS-Ausdrucks mit den ABox-Referenzen für die Diskursobjekte erfolgt durch Variablenbindung in Prolog, d.h. die Variablen **REF** aus einem FAS-Ausdruck werden in den ATL-Bäumen und -Ausdrücken übernommen und durch das BACK-System mit einer eindeutigen ABox-Referenz **uc(n)** instantiiert.

Nach den ABox-Einträgen für die Pronomen wird der aktualisierte FAS-Ausdruck in indentierter Baumnotation und die entsprechende Kette von terminalen Symbolen ausgegeben (siehe Abschnitt 3.2.5). Vor der Ausgabe wird angezeigt, unter welchem Prädikat (hier **eval_fas/2**) und mit welchem Label (hier **eval_fasn**) der Ausdruck in der Datenbasis abgespeichert wurde. Das Label **eval_fasn** wird vom MÜ-System für jedes Ergebnis eindeutig vergeben. Diese Ausgabe ist wichtig, um das Ergebnis weiterverarbeiten (siehe

Befehl `transfer/{0,1}`) oder später noch einmal anzeigen bzw. löschen zu können (siehe Befehl `pretty_print/{1,2}` bzw. `delete/{1,2}`).

Beispiellauf für die konzeptuelle Analyse eines Satzes

```
MT(g-->e)> conceptual_analysis fas2
```

```
fas(g,illoc)
--- illoc(g,lex,ass) assertion
--- clausefin(g,fin,-,+,-,-,act,event,REF)
----- fin(g,lex,+,-,-) pres_ind
----- clause(g,pred,nil,-,nil,act,event,REF)
----- v_pred(g,lex,ag_af,active,-,2,act,event,REF) erhoeihen
----- term(g,det,agent,1,3,plur,GENUS,object,REF,pron,NTYP,REAL,EXT,actor) sie
----- term(g,det,affected,3,3,plur,f,object,REF,noun,count,no,temp,no)
----- det(g,lex,3,plur,f,count,def,REF) *
----- nom(g,pred,3,plur,f,object,REF,noun,count,no,temp,no)
----- n_pred(g,lex,ag_aim,3,plur,f,object,REF,noun,count,no,temp,no)
              investition
----- term(g,det,agent,nil,3,NUM,GENUS,object,REF,pron,NTYP,REAL,EXT,
              ACT) ihr
```

```
atl(atl)
--- equal(atl,REF) erhoeihen
--- with(atl,-,agent)
----- equal(atl,REF) object
--- with(atl,-,affected)
----- equal(atl,REF) investition
----- with(atl,-,agent)
----- equal(atl,REF) object
```

```
>>> Result stored with label "atl2" under predicate "atl/2" !
```

```
>>> ABox entry without information about pronouns:
```

```
uc(694)::erhoeihen and affected: (uc(693)::investition)
```

PRONOUNS	POSSIBLE ANTECEDENTS ...			
-----	-----	-----	-----	-----
sie	usa	initiative	investitio	ihr
var	uc(689)	uc(691)	uc(693)	var
kongruenz	0	0	0	0
p_bindungen	0	0	-1000	0
naehe	40	40	-	0
subjekt	60	0	-	0
satzthema	20	0	-	0
rolle	80	0	-	80
freie_ergaenzung	0	0	-	0
diskurs	0	-1000	-	-1000

sum :	200	-960	-	-920

```
best antecedents :      usa uc(689)
```

```
actualization ...
```

PRONOUNS	POSSIBLE ANTECEDENTS ...			
-----	-----	-----	-----	-----

ihr	usa	initiative	sie	investitio
var	uc(689)	uc(691)	uc(689)	uc(693)
kongruenz	0	0	0	0
p_bindungen	0	0	0	-1000
naehe	20	20	80	-
subjekt	60	0	60	-
satzthema	20	0	20	-
rolle	80	0	80	-
freie_ergaenzung	0	0	0	-
diskurs	0	-1000	0	-

```
sum :          180      -980      240      -
```

```
best antecedents :      sie uc(689)
```

```
actualization ...
```

```
>>> ABox entries for pronouns:
```

```
uc(694)::erhoehen and agent: (uc(689)::object)
uc(693)::investition and agent: (uc(689)::object)
```

```
>>> Result stored with label "fas2" under predicate "eval_fas/2" !
```

```
fas(g,illoc)
--- illoc(g,lex,ass) assertion
--- clausefin(g,fin,-,+,-,-,act,event,uc(694))
----- fin(g,lex,+,-,-) pres_ind
----- clause(g,pred,nil,-,nil,act,event,uc(694))
----- v_pred(g,lex,ag_af,active,-,2,act,event,uc(694)) erhoehen
----- term(g,det,agent,1,3,plur,m,object,uc(689),pron,NTYP,shape,no,actor) sie
----- term(g,det,affected,3,3,plur,f,object,uc(693),noun,count,no,temp,no)
----- det(g,lex,3,plur,f,count,def,uc(693)) *
----- nom(g,pred,3,plur,f,object,uc(693),noun,count,no,temp,no)
----- n_pred(g,lex,ag_aim,3,plur,f,object,uc(693),noun,count,no,temp,no)
          investition
----- term(g,det,agent,nil,3,plur,m,object,uc(689),pron,NTYP,shape,no,
          actor) ihr
```

```
>>> ok
```

3.3.6 Transfer eines Satzes

transfer +Label	Return
-----------------	--------

Synonyme: $t/\{0,1\}$

1. *Label* ist ein Atom `eval_fasn` (Label eines gespeicherten ausgewerteten FAS-Ausdrucks der Ausgangssprache). Das Argument ist optional und wird interaktiv abgefragt, falls es nicht angegeben wird.

Mit dem Prädikat `transfer/{0,1}` wird der ausgewertete FAS-Ausdruck der Ausgangssprache (Ableitungsbaum mit FAS-Kategorien) mit dem angegebenen Label in äquivalente FAS-Ausdrücke der Zielsprache abgebildet (Transfer). Zunächst wird dabei der FAS-Ausdruck der Ausgangssprache ausgegeben oder es kommt die Meldung, daß in der Datenbasis kein FAS-Ausdruck mit dem angegebenen Label gespeichert ist.

Nach der Ausgabe des FAS-Ausdrucks wird die Kette der terminalen Symbole des entsprechenden Ausdrucks ausgegeben und es erfolgt der Transfer durch das Termersetzungungsverfahren (siehe Abschnitt 1.2.3), d.h. die Abbildung eines ausgangs- in einen oder mehrere äquivalente zielsprachliche FAS-Ausdrücke. Nach dem erfolgreichen Abschluß werden die zielsprachlichen FAS-Ausdrücke in indentierter Baumnotation und die entsprechenden Ketten von terminalen Symbolen ausgegeben (siehe Abschnitt 3.2.5). Für die Ausgabe werden die Ergebnisse beginnend mit 1 fortlaufend durchnummeriert. Vor der Ausgabe wird die Nummer eines Ergebnisses angezeigt.

Nach der Ausgabe jedes FAS-Ausdrucks wird angezeigt, unter welchem Prädikat (hier `gen/2`) und mit welchem Label (hier `genn`) der Ausdruck in der Datenbasis abgespeichert wurde. Das Label `genn` wird vom MÜ-System für jedes Ergebnis eindeutig vergeben. Diese Ausgabe ist wichtig, um das Ergebnis weiterverarbeiten (siehe Befehl `generate/{0,1}`) oder später noch einmal anzeigen bzw. löschen zu können (siehe Befehl `pretty_print/{1,2}` bzw. `delete/{1,2}`).

Anschließend wird ausgegeben, wieviele Ergebnisse der Transfer für eine Eingabestruktur erzeugen konnte und es besteht die Möglichkeit, die Ableitung eines der Ergebnisse in Einzelschritten nachzuvollziehen (siehe Abschnitt 3.3.1). Dies ist insbesondere dann wichtig, wenn die syntaktische Kontrolle nach der Termersetzung, die durch den System-Parameter `syntax` (siehe Seite 114) gesteuert wird, vorher abgeschaltet wurde, weil mit syntaktischer Kontrolle kein Ergebnis erzeugt werden konnte.

Beispiellauf für den Transfer eines Satzes

```
MT(g-->e)> transfer fas2
```

```
fas(g,illoc)
--- illoc(g,lex,ass) assertion
--- clausefin(g,fin,-,+,-,act,event,uc(694))
----- fin(g,lex,+,-) pres_ind
----- clause(g,pred,nil,-,nil,act,event,uc(694))
----- v_pred(g,lex,ag_af,active,-,2,act,event,uc(694)) erhoeuen
----- term(g,det,agent,1,3,plur,m,object,uc(689),pron,NTYP,shape,no,actor) sie
----- term(g,det,affected,3,3,plur,f,object,uc(693),noun,count,no,temp,no)
----- det(g,lex,3,plur,f,count,def,uc(693)) *
----- nom(g,pred,3,plur,f,object,uc(693),noun,count,no,temp,no)
----- n_pred(g,lex,ag_aim,3,plur,f,object,uc(693),noun,count,no,temp,no)
          investition
----- term(g,det,agent,nil,3,plur,m,object,uc(689),pron,NTYP,shape,no,
          actor) ihr
```

```
assertion pres ind erhoeuen sie * investition ihr
```

```
>>> Rewrite result no 1
```

```
>>> Result stored with label "gen2" under predicate "gen/2" !
```

```
fas(e,illoc)
--- illoc(e,lex,ass) assertion
--- clausefin(e,fin,-,-,+,-,act,event,uc(694))
----- fin(e,lex,+,-) pres_ind
----- clause(e,pred,nil,-,-,nil,act,event,uc(694))
----- v_pred(e,lex,ag_af,active,-,-,2,act,event,uc(694)) increase
----- term(e,det,agent,1,3,plur,n,object,uc(689),pron,count,shape,no,actor)
          persprn
----- term(e,det,affected,3,3,plur,n,object,uc(693),noun,count,no,temp,no)
```

```

----- det(e,lex,3,plur,n,count,def,uc(693)) *
----- nom(e,pred,3,plur,n,object,uc(693),noun,count,no,temp,no)
----- n_pred(e,lex,ag_aim,3,plur,n,object,uc(693),noun,count,no,temp,no)
           investment
----- term(e,det,agent,nil,3,plur,n,object,uc(689),pron,count,shape,no,
           actor) possprn

assertion pres ind increase persprn * investment possprn

>>> 1 rewrite result(s) !
>>> Do you want to trace one of the rewrite results ?
>>> Input: y(es) [RETURN] or n(o) [RETURN] > n

>>> ok

```

3.3.7 Generierung eines Satzes

generate +Label	Return
-----------------	--------

Synonyme: $g/\{0,1\}$, $gen/\{0,1\}$, $generator/\{0,1\}$, $generating/\{0,1\}$

1. *Label* ist ein Atom *genn* (Label eines gespeicherten FAS-Ausdrucks der Zielsprache). Das Argument ist optional und wird, falls nicht angegeben, interaktiv abgefragt.

Mit dem Prädikat $generate/\{0,1\}$ werden syntaktische Strukturen der Zielsprache und entsprechende Sätze aus dem FAS-Ausdruck der Zielsprache (Ableitungsbaum mit FAS-Kategorien) mit dem angegebenen Label erzeugt (Generierung). Zunächst wird dabei der FAS-Ausdruck der Zielsprache ausgegeben oder es kommt die Meldung, daß in der Datenbasis kein FAS-Ausdruck mit dem angegebenen Label gespeichert ist.

Nach der Ausgabe des FAS-Ausdrucks wird die Kette der terminalen Symbole des entsprechenden Ausdrucks ausgegeben und es erfolgt die Generierung durch das Termersetzungsverfahren (siehe Abschnitt 1.2.3), d.h. die Abbildung eines zielsprachlichen FAS-Ausdrucks in eine oder mehrere äquivalente zielsprachliche syntaktische Strukturen (Ableitungsbäume mit komplexen Kategorien). Nach dem erfolgreichen Abschluß werden die zielsprachlichen Strukturen in indentierter Baumnotation und die entsprechenden Ketten von terminalen Symbolen¹¹ ausgegeben (siehe Abschnitt 3.2.5). Für die Ausgabe werden die Ergebnisse beginnend mit 1 fortlaufend durchnummeriert. Vor der Ausgabe wird die Nummer eines Ergebnisses angezeigt.

Nach der Ausgabe jeder syntaktischen Struktur wird angezeigt, unter welchem Prädikat (hier *res/2*) und mit welchem Label (hier *resn*) die Struktur in der Datenbasis abgespeichert wurde. Das Label *resn* wird vom MÜ-System für jedes Ergebnis eindeutig vergeben. Diese Ausgabe ist wichtig, um das Ergebnis später noch einmal anzeigen bzw. löschen zu können (siehe Befehl *pretty_print/\{1,2\}* bzw. *delete/\{1,2\}*).

Anschließend wird ausgegeben, wieviele Ergebnisse die Generierung für eine Eingabestruktur erzeugen konnte und es besteht die Möglichkeit, die Ableitung eines der Ergebnisse in Einzelschritten nachzuvollziehen (siehe Abschnitt 3.3.1). Dies ist insbesondere dann wichtig, wenn die syntaktische Kontrolle nach der Termersetzung, die durch den System-Parameter *syntax* (siehe Seite 114) gesteuert wird, vorher abgeschaltet wurde, weil mit syntaktischer Kontrolle kein Ergebnis erzeugt werden konnte.

¹¹Aus den prä-terminalen und terminalen syntaktischen Kategorien werden die Wortformen der Zielsprache morphologisch synthetisiert, bevor sie ausgegeben werden.

Beispiellauf für die Generierung eines Satzes

```
MT(g-->e)> generate gen2
```

```
fas(e,illoc)
--- illoc(e,lex,ass) assertion
--- clausefin(e,fin,-,-,+,-,act,event,uc(694))
----- fin(e,lex,+,-,-) pres_ind
----- clause(e,pred,nil,-,-,nil,act,event,uc(694))
----- v_pred(e,lex,ag_af,active,-,-,2,act,event,uc(694)) increase
----- term(e,det,agent,1,3,plur,n,object,uc(689),pron,count,shape,no,actor)
          persprn
----- term(e,det,affected,3,3,plur,n,object,uc(693),noun,count,no,temp,no)
----- det(e,lex,3,plur,n,count,def,uc(693)) *
----- nom(e,pred,3,plur,n,object,uc(693),noun,count,no,temp,no)
----- n_pred(e,lex,ag_aim,3,plur,n,object,uc(693),noun,count,no,temp,no)
          investment
----- term(e,det,agent,nil,3,plur,n,object,uc(689),pron,count,shape,no,
          actor) possprn
```

```
assertion pres ind increase persprn * investment possprn
```

```
>>> Rewrite result no 1
```

```
>>> Result stored with label "res2" under predicate "res/2" !
```

```
v(gpsg,2,^,3,plur,^,+,-,^,-,fin,+,active,^,-,-,^)
--- d(gpsg,2,^,3,plur,n,nom,top,-)
----- d(gpsg,1,^,3,plur,n,nom,^,^)
----- d(gpsg,0,1,3,plur,n,nom,^,^) persprn
--- v(gpsg,2,^,3,plur,^,-,-,^,^,fin,+,active,^,-,-,^)
----- v(gpsg,0,2,3,plur,^,-,-,^,^,fin,+,active,nil,-,^,^) increase
----- d(gpsg,2,^,3,plur,n,acc,^,-)
----- d(gpsg,1,^,3,plur,n,acc,^,^)
----- n(gpsg,2,^,3,plur,n,acc,^)
----- n(gpsg,1,^,3,plur,n,acc,^)
----- n(gpsg,0,4,3,plur,n,acc,^) investment
----- p(gpsg,2,^,of,-,^)
----- p(gpsg,0,^,of,^,^) of
----- d(gpsg,2,^,3,plur,n,acc,^,-)
----- d(gpsg,1,^,3,plur,n,acc,^,^)
----- d(gpsg,0,1,3,plur,n,acc,^,^) possprn
```

```
they increase investments of them
```

```
>>> Rewrite result no 2
```

```
>>> Result stored with label "res3" under predicate "res/2" !
```

```
v(gpsg,2,^,3,plur,^,+,-,^,-,fin,+,active,^,-,-,^)
--- d(gpsg,2,^,3,plur,n,nom,top,-)
----- d(gpsg,1,^,3,plur,n,nom,^,^)
----- d(gpsg,0,1,3,plur,n,nom,^,^) persprn
--- v(gpsg,2,^,3,plur,^,-,-,^,^,fin,+,active,^,-,-,^)
----- v(gpsg,0,2,3,plur,^,-,-,^,^,fin,+,active,nil,-,^,^) increase
----- d(gpsg,2,^,3,plur,n,acc,^,-)
----- d(gpsg,1,^,3,plur,n,acc,^,^)
----- d(gpsg,0,2,3,plur,n,acc,^,^) their
----- n(gpsg,2,^,3,plur,n,acc,^)
----- n(gpsg,1,^,3,plur,n,acc,^)
----- n(gpsg,0,4,3,plur,n,acc,^) investment
```

they increase their investments

```
>>> 2 rewrite result(s) !
>>> Do you want to trace one of the rewrite results ?
>>> Input: y(es) [RETURN] or n(o) [RETURN] > n

>>> ok
```

3.3.8 Morphologische Synthese einer Wortform

morphological_synthesis +Hauptkategorie +Spezifikationen +Wortstamm	Return
---	--------

Synonyme: ms/3

1. *Hauptkategorie* ist ein Atom (Hauptkategorie der Syntax der Zielsprache).
2. *Spezifikationen* ist eine Liste mit Merkmalspezifikationen der Form *+Merkmal* : *+Wert* für die Hauptkategorie mit
 - (a) *Merkmal* ist ein Atom (syntaktisches Merkmal).
 - (b) *Wert* ist ein Atom oder eine ganze Zahl (Wert des Merkmals).
3. *Wortstamm* ist ein Atom (Wortstamm der Zielsprache).

Mit dem Prädikat `morphological_synthesis/3` wird aus einem Wortstamm und einer syntaktischen komplexen Kategorie der Zielsprache die entsprechende Wortform generiert und auf dem Bildschirm ausgegeben. Diese Komponente basiert auf dem SUTRA-System (siehe Abschnitt 1.2.1). Die komplexe syntaktische Kategorie der Zielsprache wird durch ihre Hauptkategorie und einer zugehörigen Liste von Merkmalspezifikationen angegeben.

Beispiellauf für die morphologische Synthese einer Wortform

```
MT(g-->e)> morphological_synthesis v [bar:0,pers:3,num:sing,neg:-] take
takes

>>> ok
```

3.3.9 Übersetzung eines Satzes

translate	Return
-----------	--------

Synonyme: a/0, translation/0, mt/0

Mit dem Prädikat `translate/0` wird ein einzugebender Satz der Ausgangssprache in die Zielsprache übersetzt. Zunächst erfolgt dabei die Eingabe eines Satzes der Ausgangssprache, die mit `Return` beendet wird. Danach wird der eingegebene Satz morphologisch analysiert. Dabei werden die Lesarten der Wortformen, die im Satz vorkommen, auf dem Bildschirm ausgegeben. Anschließend wird der Satz dann nacheinander syntaktisch, semantisch und konzeptuell analysiert. Nach der Interpretation der anaphorischen Bezüge werden der Transfer und die Generierung aufgerufen. Im Anschluß an eine erfolgreiche Generierung wird der zielsprachliche Satz ausgegeben. Der Aufruf einer Komponente wird auf dem Bildschirm angezeigt. Wenn durch die Komponente ein Ergebnis erzeugt werden konnte, wird angezeigt, unter welchem Prädikat (hier `gpsg/2`,

fas/2, *atl/2*, *eval_fas/2*, *gen/2* bzw. *res/2*) und mit welchem Label (hier *gpsgn*, *fasn*, *atln*, *eval_fasn*, *genn* bzw. *resn*) das Ergebnis in der Datenbasis abgespeichert wurde. Alle Label werden vom MÜ-System für jedes Ergebnis eindeutig vergeben. Diese Ausgabe ist wichtig, um das Ergebnis später noch einmal anzeigen bzw. löschen zu können (siehe Befehl `pretty_print/{1,2}` bzw. `delete/{1,2}`).

In jedem Strukturabbildungsschritt werden sämtliche mögliche Ergebnisse erzeugt. Eine Desambiguierung dieser Ergebnisse erfolgt außer durch die Anapherninterpretation nach der konzeptuellen Analyse nicht. Wenn eine Komponente eine Ergebnisstruktur erzeugen konnte, wird diese zunächst an die nachfolgende Komponente zur Verarbeitung weitergereicht. Erst wenn eine Komponente erfolglos ist oder ein Satz erfolgreich übersetzt werden konnte, wird die Erzeugung von alternativen Lösungen der vorangehenden Komponente durch Backtracking eingeleitet, bis keine Komponente mehr weitere Lösungen erzeugen kann.

Kommen in einem ausgangssprachlichen Satz Personal- oder Possessivpronomina vor, wird durch die Anapherninterpretationskomponente eine Auswertungstabelle ausgegeben (siehe Seite 85).

Beispiellauf für die Übersetzung eines Satzes

```
MT(g-->e)> translate
>>> Please input a sentence ending with "[RETURN]" !
```

```
Input> die usa ergreifen neue initiativen
```

```
>>> Lemmatization for "die".
d(1,3,-,2,nom,1)rel[3,-,2,r]
d(1,3,-,2,acc,1)rel[3,-,2,r]
d(1,3+,GEN0,nom,1)rel[3+,GEN0,r]
d(1,3+,GEN0,acc,1)rel[3+,GEN0,r]
d(2,3,-,2,nom,1)
d(2,3,-,2,acc,1)
d(2,3+,GEN0,nom,1)
d(2,3+,GEN0,acc,1)
```

```
>>> Lemmatization for "usa".
n(1+,1,CAS0)
```

```
>>> Lemmatization for "ergreifen".
v(6,PER0,PLU0,THEM0,-,^,-,inf,-) inf
v(6,1+,^,-,^,ACO,MC0,fin,-) pres
v(6,3+,^,-,^,ACO,MC0,fin,-) pres
v(6,1+,^,-,^,ACO,MC0,fin,-) kon1
v(6,3+,^,-,^,ACO,MC0,fin,-) kon1
```

```
>>> Lemmatization for "neue".
a(1,3,-,2,acc,FLEX0)
a(1,3,-,2,nom,FLEX0)
a(1,3,-,1,nom,1)
a(1,3,-,3,nom,1)
a(1,3,-,3,acc,1)
a(1,3+,GEN0,nom,3)
a(1,3+,GEN0,acc,3)
```

```
>>> Lemmatization for "initiativen".
n(5+,2,CAS0)
```

```

-----
>>> Input: die usa ergreifen neue initiativen

>>> Parsing ...
>>> Result stored with label "gpsg0" under predicate "gpsg/2" !
>>> done
>>> Mapping S-GPSG onto S-FAS ...
>>> Result stored with label "fas0" under predicate "fas/2" !
>>> done
>>> Evaluating FAS ...
>>> Result stored with label "atl0" under predicate "atl/2" !
>>> Result stored with label "fas0" under predicate "eval_fas/2" !
>>> done
>>> Transferring S-FAS onto T-FAS ...
>>> Result stored with label "gen0" under predicate "gen/2" !
>>> done
>>> Generation ...
>>> Result stored with label "res0" under predicate "res/2" !
>>> done

```

the usa new initiatives take

```

>>> Trying new generation ...
>>> no result
>>> Trying new transfer from S-FAS onto T-FAS ...
>>> no result
>>> Trying new evaluation ...
>>> no result
>>> Trying new map from S-GPSG onto S-FAS ...
>>> no result
>>> Trying new parse ...
>>> Result stored with label "gpsg1" under predicate "gpsg/2" !
>>> done
>>> Mapping S-GPSG onto S-FAS ...
>>> Result stored with label "fas1" under predicate "fas/2" !
>>> done
>>> Evaluating FAS ...
>>> Result stored with label "atl1" under predicate "atl/2" !
>>> Result stored with label "fas1" under predicate "eval_fas/2" !
>>> done
>>> Transferring S-FAS onto T-FAS ...
>>> Result stored with label "gen1" under predicate "gen/2" !
>>> done
>>> Generation ...
>>> Result stored with label "res1" under predicate "res/2" !
>>> done

```

the usa take new initiatives

```

>>> Trying new generation ...
>>> no result
>>> Trying new transfer from S-FAS onto T-FAS ...
>>> no result
>>> Trying new evaluation ...
>>> no result
>>> Trying new map from S-GPSG onto S-FAS ...
>>> no result
>>> Trying new parse ...
>>> no result

```

```
>>> No (more) translations !
```

```
>>> ok
```

3.4 Aktionen mit gespeicherten Repräsentationen

In diesem Abschnitt werden die MÜ-System-Befehle vorgestellt, mit denen die Repräsentationen, die durch die einzelnen Übersetzungsschritte (siehe Abschnitt 3.3) erzeugt und in der Datenbasis gespeichert werden, ausgegeben, gelöscht oder in einer Datei gespeichert werden können.

3.4.1 Ausgabe von Repräsentationen aus der Datenbasis

<code>pretty_print +Prädikat +Label</code>	<code>Return</code>
--	---------------------

Synonyme: `p/{1,2}`

1. *Prädikat* ist ein Atom `gpsg`, `fas`, `atl`, `eval_fas`, `gen` oder `res` (Name eines 2-stelligen Prädikats, unter dem die Repräsentationen in der Datenbasis gespeichert werden). Das Argument ist optional.
2. *Label* ist ein Atom `gpsgn`, `fasn`, `atln`, `eval_fasn`, `genn` oder `resn` (Label einer Repräsentation, die in der Datenbasis gespeichert ist).

Mit dem Prädikat `pretty_print/{1,2}` wird eine Repräsentation, die von einer Abbildungskomponente unter dem angegebenen Prädikat und Label in der Datenbasis abgespeichert wird (siehe Abschnitt 3.3), in indenterter Baumnotation mit komplexen Kategorien ausgegeben (siehe Abschnitt 3.2.5). Falls sich keine Struktur mit dem angegebenen Label in der Datenbasis findet, wird eine entsprechende Meldung gemacht.

3.4.2 Löschen von Repräsentationen aus der Datenbasis

<code>delete +Prädikat +Label</code>	<code>Return</code>
--------------------------------------	---------------------

Synonyme: `d/{0,1}`

1. *Prädikat* ist ein Atom `gpsg`, `fas`, `atl`, `eval_fas`, `gen` oder `res` (Name eines 2-stelligen Prädikats, unter dem die Repräsentationen in der Datenbasis gespeichert werden). Das Argument ist optional.
2. *Label* ist ein Atom `gpsgn`, `fasn`, `atln`, `eval_fasn`, `genn` oder `resn` (Label einer Repräsentation, die in der Datenbasis gespeichert ist).

Mit dem Prädikat `delete/{0,1}` wird eine Repräsentation, die von einer Abbildungskomponente unter dem angegebenen Prädikat und Label in der Datenbasis gespeichert wird (siehe Abschnitt 3.3), unwiederbringlich gelöscht. Falls sich keine Struktur mit dem angegebenen Label in der Datenbasis findet, wird eine entsprechende Meldung gemacht. Nach dem Löschen erfolgt keine besondere Meldung.

3.4.3 Schreiben gespeicherter Repräsentationen in eine Datei

<code>file +Prädikat</code>	<code>Return</code>
-----------------------------	---------------------

Synonyme: keine definiert

1. *Prädikat* ist ein Atom `gpsg`, `fas`, `atl`, `eval_fas`, `gen` oder `res` (Name eines 2-stelligen Prädikats, unter dem die Repräsentationen in der Datenbasis gespeichert werden).

Mit dem Prädikat `file/1` werden die Repräsentationen, die von einer Abbildungskomponente unter dem angegebenen Prädikat in der Datenbasis gespeichert sind (siehe Abschnitt 3.3), in eine Datei geschrieben. Diese Datei kann mit dem Befehl `load_data/1` wieder geladen werden (siehe Seite 71). Nach der Eingabe des Befehls fordert der Interpretierer zur Eingabe des Dateinamens auf. Die Eingabe muß mit `Return` abgeschlossen werden. Anschließend wird berichtet, ob das Beschreiben der Datei erfolgreich war. Ein Fehler kann beispielsweise auftreten, wenn keine Struktur mit dem angegebenen Prädikat in der Datenbasis gespeichert ist.

3.5 Laden von linguistischen Daten

In der Datei `mtmenu.pl` (siehe Abschnitt 3.5.1 und Anhang B) sind die Quelldateien für die linguistischen Daten für ein Fragment des Deutschen (Sprachbezeichner `g`) und seiner Entsprechung im Englischen (Sprachbezeichner `e`) spezifiziert. Allerdings ist mit den vordefinierten Daten nur die Übersetzungsrichtung Deutsch → Englisch möglich.

Zum Laden der Daten kann festgelegt werden, ob die Dateien, die in der Datei `mtmenu.pl` spezifiziert sind (Defaultdaten), geladen oder ob die Dateinamen manuell eingegeben werden sollen. Um dies festzulegen, wird der System-Parameter `default` auf `on` bzw. `off` gesetzt (siehe Seite 107). Vor dem Laden der linguistischen Daten wird interaktiv abgefragt, wie dieser Parameter gesetzt werden soll.

Es gibt Laderoutinen zum Laden sämtlicher für das MÜ-System notwendigen linguistischen Daten oder zum Laden der linguistischen Daten, die für eine einzelne Systemkomponente benötigt werden. Die Laderoutinen für die einzelnen Komponenten dienen im wesentlichen dazu, die Daten für eine Komponente nach Änderungen oder Erweiterungen neu zu laden, ohne die unveränderten Daten noch einmal neu laden zu müssen. Alle anderen Daten bleiben unberührt, so daß die vollständige Übersetzung eines Satzes nicht mehr möglich ist, wenn keine geeigneten Schnittstellen von den neu geladenen zu den übrigen Daten existieren. Dies ist in der Regel der Fall, wenn das MÜ-System als Testumgebung für die Entwicklung von linguistischen Daten einer einzelnen Komponente eingesetzt wird. Beim Laden jeder Datei mit linguistischen Daten wird folgendermaßen vorgegangen:

1. Falls der System-Parameter `default` auf `on` gesetzt ist, wird die Datei zunächst in dem Verzeichnis, das im vierten Argument des Faktums `gram_file/5` (siehe Seite 98) angegeben ist, gesucht. Das Verzeichnis muß nach dem Pfad existieren, der im zweiten Argument des Faktums `language_directory/2` (siehe Seite 98) angegeben ist. Existiert der Pfad, das Verzeichnis oder die Datei nicht, wird die Datei im Systemverzeichnis `~/FAST-v94/prog/mt` gesucht. Existiert die Datei dort ebenfalls nicht, wird sie interaktiv abgefragt (siehe Punkt 2).

Ist anstelle des Dateinamens im Faktum `gram_file/5` der Wert `dummy` spezifiziert (bei den TBox-Dateien `[]`), so wird der entsprechende Teil der Laderoutine übersprungen und es werden keine neuen Daten geladen.

2. Falls der System-Parameter `default` auf `off` gesetzt ist oder falls die Datei, die im Faktum `gram_file/5` spezifiziert ist, nicht existiert (siehe Punkt 1), wird der Dateiname interaktiv abgefragt. Die Datei mit dem eingegebenen Namen wird zunächst in dem Verzeichnis gesucht, das im vierten Argument des Faktums `gram_file/5` angegeben ist. Das Verzeichnis muß nach dem Pfad existieren, der im zweiten Argument des Faktums `language_directory/2` angegeben ist. Existiert das Faktum `gram_file/5`, der Pfad, das Verzeichnis oder die Datei nicht, wird die Datei im Systemverzeichnis `~/FAST-v94/prog/mt` gesucht.

Wird anstelle des Dateinamens nur `Return` eingegeben, so wird der entsprechende Teil der Laderoutine übersprungen und es werden keine neuen Daten geladen.

Es besteht die Möglichkeit, linguistische Daten für andere Sprachen bzw. Sprachpaare zu entwickeln. Zu diesem Zweck wurden spezielle Compiler bzw. Editoren entwickelt, die eine benutzerfreundliche Eingabe der Daten ermöglicht (vgl. [Dunker 93] und [Weisweber 94b]). Wenn die Dateien, in denen die neu entwickelten Daten enthalten sind, in der Datei `mtmenu.pl` unter dem entsprechenden Sprachbezeichner spezifiziert (siehe Abschnitt 3.5.1 und Anhang B) oder manuell eingegeben werden, kommen auch andere Übersetzungsrichtungen als Deutsch → Englisch in Frage.

3.5.1 Die Datei `mtmenu.pl`

In der Datei `mtmenu.pl` im Verzeichnis `~/FAST-v94/prog/mt` sind Namen für Verzeichnisse und Dateien spezifiziert, die linguistische Daten und die Online-Dokumentation der MÜ-System-Befehle enthalten. Die vordefinierte Datei `mtmenu.pl` ist im Abschnitt B angegeben.

Das Laden der linguistischen Daten hängt vom Wert des System-Parameters `default` ab (siehe Seite 107). Ist er auf `on` gesetzt, versuchen die Laderoutinen des MÜ-Systems zunächst, die in `mtmenu.pl` spezifizierten Dateien mit den linguistischen Daten zu laden. Erst wenn diese nicht existieren oder der System-Parameter `default` auf `off` gesetzt ist, werden die Dateinamen interaktiv abgefragt.

Das Verzeichnis, das die Dateien mit der Online-Dokumentation der MÜ-System-Befehle in Englisch enthält, wird in der Datei `mtmenu.pl` (siehe Abschnitt 3.5.1 und Anhang B) mithilfe des Prädikats `help_directory/1` spezifiziert.

`help_directory(-Pfad)`

1. *Pfad* ist ein Atom (Verzeichnis, in dem die Dateien mit der Online-Dokumentation der MÜ-System-Befehle enthalten sind).

Die Verzeichnisse und Dateien mit den linguistischen Daten werden in der Datei `mtmenu.pl` (siehe Abschnitt 3.5.1 und Anhang B) mithilfe der beiden Prädikate `language_directory/2` bzw. `gram_file/5` spezifiziert.

`language_directory(+Bezeichner, -Pfad)`

1. *Bezeichner* ist ein Atom bzw. ein Term *Ausgangssprachbezeichner* $\rightarrow$ *Zielsprachbezeichner* (*AS-Bezeichner* $\rightarrow$ *ZS-Bezeichner*). Ein Atom indiziert eine Sprache und ein Term mit dem Prädikat $\rightarrow/2$ eine Übersetzungsrichtung. Für die linguistischen Daten jeder Sprache und für die Transferregeln sollte ein eigenes Verzeichnis angelegt werden. Der spezifizierte Bezeichner ist das Argument für die Laderoutinen (siehe Abschnitte 3.5.3 bis 3.5.9).
2. *Pfad* ist ein Atom (Pfad, nach dem sämtliche linguistischen Daten der entsprechenden Sprache spezifiziert sind).

`gram_file(+Bezeichner, +Kürzel, -Kommentar, -Verzeichnis, -Dateiname)`

1. *Bezeichner* siehe erstes Argument von `language_directory/2`.
2. *Kürzel* ist ein Atom, das die Art der Daten indiziert, die in der spezifizierten Datei enthalten sind. Die Prädikate, die in den entsprechenden Dateien definiert sein müssen, werden im Abschnitt 2.4 vorgestellt. Dabei bedeuten

Morphologische Analyse bzw. Synthese

<code>glex</code>	Abbildungslexikon GPSG $\rightarrow$ FAS
<code>flex</code>	Abbildungslexikon FAS $\rightarrow$ TBox
<code>mlex</code>	Morphologisches Lexikon
<code>mdata</code>	Flexionsparadigmen

Parsing (GPSG)

<code>alias</code>	Abkürzungen für die Ausgabe von GPSG-Kategorien
<code>fcr</code>	FCRs
<code>id</code>	ID-Regeln und Metaregeln
<code>idc</code>	Kommentare zu den ID-Regeln
<code>lp</code>	LP-Aussagen
<code>dom</code>	Dominanzrelation aus den ID-Regeln

Semantische Analyse bzw. Generierung

<code>gdef</code>	Spezifikation für GPSG-Kategorienterme
<code>hdef</code>	Spezifikation für hybride (syn. und sem.) Kategorienterme
<code>fdef</code>	Spezifikation für FAS-Kategorienterme
<code>fas</code>	FAS-Grammatik
<code>idgen</code>	Kontextfreie Grammatik für die Zielsprache
<code>flex</code>	Abbildungslexikon FAS $\rightarrow$ TBox
<code>conf</code>	Kasus-Rollen-Tabellen
<code>map</code>	Termersetzungssystem für die semantische Analyse
<code>pa</code>	Termersetzungssystem für die Generierung

Konzeptuelle Analyse

fdef	Spezifikation für FAS-Kategorienterme
ade	Spezifikation für ATL-Kategorienterme
atgram	ATL-Grammatik
atllex	ATL-Lexikon
fasatl	Termersetzungssystem für die konzeptuelle Analyse

Anapherninterpretation

pront	Pronomentabelle
edefs	Parameter für die Anapherninterpretation

Transfer

fdef	Spezifikation für FAS-Kategorienterme (AS und ZS)
fas	FAS-Grammatik
flex	Abbildungslexikon $FAS \rightarrow TBox$ (ZS)
trule	Termersetzungssystem für den Transfer

BACK-System

tbox	Prolog-Liste der Dateien mit den TBox-Definitionen
-------------	--

3. *Kommentar* ist ein Atom, das die in der Datei enthaltenen Daten in natürlicher Sprache näher beschreibt. Dieser Kommentar wird beim Laden ausgegeben, wenn der System-Parameter **default** auf **on** gesetzt ist (siehe Seite 107).
4. *Verzeichnis* ist ein Atom, welches das Verzeichnis angibt, in dem die Datei existiert. Das Verzeichnis muß nach dem Pfad existieren, der für den Sprachbezeichner mithilfe des Prädikats **language_directory/2** angegeben wurde.
5. *Dateiname* ist ein Atom, das den Namen für eine Datei mit den durch das Kürzel indizierten Daten angibt. Wird keine Extension angegeben, so wird zunächst versucht, die Datei mit der Extension **.pl** zu laden. Falls diese nicht existiert, wird die Extension **.int** probiert. Wenn dies ebenfalls nicht möglich ist, wird das Atom selbst als Dateiname betrachtet.

Wird anstatt eines Dateinames das Atom **dummy** angegeben, so werden keine linguistischen Daten für das Kürzel geladen. Das gleiche geschieht, wenn unter dem Kürzel **tbox** die leere Liste statt einer Liste von Dateien mit TBox-Definitionen spezifiziert wird.

3.5.2 Laden sämtlicher linguistischen Daten

data	Return
-------------	---------------

Synonyme: keine definiert

Mit dem Prädikat **data/0** werden sämtliche linguistischen Daten des MÜ-Systems neu geladen. Zum Laden der Daten müssen zunächst Ausgangs- und Zielsprache (AS

bzw. ZS) für das System spezifiziert werden. Die erforderliche Eingabe erfolgt durch `AS-Bezeichner - ZS-Bezeichner` `Return`. Wie oben bereits erwähnt, sind nur für ein kleines Fragment der Übersetzungsrichtung Deutsch → Englisch, Daten vordefiniert (siehe Anhang A), d.h. nach dem Aufruf sollte `g-e` `Return` eingegeben werden.

Anschließend wird abgefragt, wie der Parameter `default` (siehe Seite 107) für den Ladevorgang gesetzt werden soll. Wird die Frage mit `yes` `Return` beantwortet, so wird dieser Parameter auf `on` gesetzt und die Dateien, die in der Datei `mtmenu.pl` spezifiziert sind (siehe Abschnitt 3.5.1 und Anhang B), werden ohne weitere Interaktion mit dem Benutzer geladen, es sei denn, eine der Dateien existiert nicht. Wird die Frage mit `no` `Return` beantwortet, so wird der Parameter `default` auf `off` gesetzt und sämtliche Dateien für die linguistischen Daten werden interaktiv abgefragt. Der Parameter `default` behält nach dem Laden den in der Laderoutine angegebenen Wert.

Im nächsten Schritt werden zunächst das MÜ- und das BACK-System initialisiert. Anschließend werden die linguistischen Daten für die einzelnen Systemkomponenten in der folgenden Reihenfolge geladen:

1. syntaktische Analyse (siehe Abschnitt 3.5.3),
2. semantische Analyse (siehe Abschnitt 3.5.4),
3. konzeptuelle Analyse (siehe Abschnitt 3.5.5),
4. Anaphernauswertung (siehe Abschnitt 3.5.6),
5. Textrepräsentation (siehe Abschnitt 3.5.7),
6. Transfer (siehe Abschnitt 3.5.8) und
7. Generierung (siehe Abschnitt 3.5.9).

3.5.3 Laden der Daten für die syntaktische Analyse

`load_parser +Sprachbezeichner` `Return`

Synonyme: `lp/1`

1. *Sprachbezeichner* ist ein Atom (das erste Argument der Fakten mit den Prädikaten `language_directory/2` und `gram_file/5`, siehe Seite 98).

Mit dem Prädikat `load_parser/1` werden die linguistischen Daten, die für die syntaktische Analyse benötigt werden, neu in das MÜ-System geladen. Alle anderen Daten bleiben unverändert, so daß die vollständige Übersetzung eines Satzes nicht mehr möglich ist, wenn keine geeigneten Schnittstellen von der syntaktischen zur semantischen Analyse existieren.

Anschließend wird abgefragt, wie der Parameter `default` (siehe Seite 107) für den Ladevorgang gesetzt werden soll. Wird die Frage mit `yes` `Return` beantwortet, so wird dieser Parameter auf `on` gesetzt und die Dateien, die in der Datei `mtmenu.pl` spezifiziert sind (siehe Abschnitt 3.5.1 und Anhang B), werden ohne weitere Interaktion mit dem Benutzer geladen, die Existenz sämtlicher Daten vorausgesetzt. Wird die Frage mit `no` `Return` beantwortet, so wird der Parameter `default` auf `off` gesetzt und die linguistischen Daten für die syntaktische Analyse werden interaktiv abgefragt. Der Parameter

default behält nach dem Laden den in der Laderoutine angegebenen Wert. Anschließend werden die linguistischen Daten für die syntaktische Analyse in der folgenden Reihenfolge geladen:

1. Abkürzungen für die Ausgabe von GPSG-Kategorien (Alias),
2. FCRs,
3. ID-Regeln,
4. Kommentare zu den ID-Regeln,
5. LP-Aussagen,
6. Dominanzrelation aus den ID-Regeln,
7. Abbildungslexikon GPSG $\rightarrow$ FAS,
8. Abbildungslexikon FAS $\rightarrow$ TBox,
9. morphologisches Lexikon und
10. Flexionsparadigmen.

3.5.4 Laden der Daten für die semantische Analyse

<code>load_semantic_analysis +Sprachbezeichner</code>	<code>Return</code>
---	---------------------

Synonyme: lsa/1

1. *Sprachbezeichner* ist ein Atom (das erste Argument der Fakten mit den Prädikaten `language_directory/2` und `gram_file/5`, siehe Seite 98).

Mit dem Prädikat `load_semantic_analysis/1` werden die linguistischen Daten, die für die semantische Analyse benötigt werden, neu in das MÜ-System geladen. Alle anderen Daten bleiben unverändert, so daß die vollständige Übersetzung eines Satzes nicht mehr möglich ist, wenn keine geeigneten Schnittstellen von der semantischen Analyse zur syntaktischen und konzeptuellen Analyse und zum Transfer existieren.

Anschließend wird abgefragt, wie der Parameter **default** (siehe Seite 107) für den Ladevorgang gesetzt werden soll. Wird die Frage mit

yes

Return

 beantwortet, so wird dieser Parameter auf **on** gesetzt und die Dateien, die in der Datei `mtmenu.pl` spezifiziert sind (siehe Abschnitt 3.5.1 und Anhang B), werden ohne weitere Interaktion mit dem Benutzer geladen, es sei denn, daß eine der Dateien nicht existiert. Wird die Frage mit

no

Return

 beantwortet, so wird der Parameter **default** auf **off** gesetzt und die linguistischen Daten für die semantische Analyse werden interaktiv abgefragt. Der Parameter **default** behält nach dem Laden den in der Laderoutine angegebenen Wert. Anschließend werden die linguistischen Daten für die semantische Analyse in der folgenden Reihenfolge geladen:

1. Spezifikation für GPSG-Kategorienterme,
2. Spezifikation für hybride Kategorienterme,¹²

¹²Hybride Kategorien enthalten sowohl syntaktische als auch semantische Informationen.

3. Spezifikation für FAS-Kategorienterme,
4. FAS-Grammatik,
5. Abbildungslexikon FAS → TBox,
6. Kasus-Rollen-Tabellen und
7. Termersetzungssystem für die Abbildung von GPSG-Strukturen in FAS-Ausdrücke.

3.5.5 Laden der Daten für die konzeptuelle Analyse

`load_conceptual_analysis +Sprachbezeichner` `Return`

Synonyme: lca/1

1. *Sprachbezeichner* ist ein Atom (das erste Argument der Fakten mit den Prädikaten `language_directory/2` und `gram_file/5`, siehe Seite 98).

Mit dem Prädikat `load_conceptual_analysis/1` werden die linguistischen Daten, die für die konzeptuelle Analyse benötigt werden, neu in das MÜ-System geladen. Alle anderen Daten bleiben unverändert, so daß die vollständige Übersetzung eines Satzes nicht mehr möglich ist, wenn keine geeigneten Schnittstellen von der konzeptuellen Analyse zur semantischen Analyse und zur Anapherninterpretation existieren.

Anschließend wird abgefragt, wie der Parameter `default` (siehe Seite 107) für den Ladevorgang gesetzt werden soll. Wird die Frage mit `yes` `Return` beantwortet, so wird dieser Parameter auf `on` gesetzt und die Dateien, die in der Datei `mtmenu.pl` spezifiziert sind (siehe Abschnitt 3.5.1 und Anhang B), werden ohne weitere Interaktion mit dem Benutzer geladen, sofern alle Dateien vorhanden sind. Wird die Frage mit `no` `Return` beantwortet, so wird der Parameter `default` auf `off` gesetzt und die linguistischen Daten für die konzeptuelle Analyse werden interaktiv abgefragt. Der Parameter `default` behält nach dem Laden den in der Laderoutine angegebenen Wert. Anschließend werden die linguistischen Daten für die konzeptuelle Analyse in der folgenden Reihenfolge geladen:

1. Spezifikation für FAS-Kategorienterme,
2. Spezifikation für ATL-Kategorienterme,
3. ATL-Grammatik,
4. ATL-Lexikon und
5. Termersetzungssystem für die Abbildung von FAS-Ausdrücken in ATL-Strukturen.

3.5.6 Laden der Daten für die Anapherninterpretation

`load_anaphora_evaluation +Sprachbezeichner` `Return`

Synonyme: lae/1

1. *Sprachbezeichner* ist ein Atom (das erste Argument der Fakten mit den Prädikaten `language_directory/2` und `gram_file/5`, siehe Seite 98).

Mit dem Prädikat `load_anaphora_evaluation/1` werden die linguistischen Daten, die für die Anapherninterpretation benötigt werden, neu in das MÜ-System geladen. Alle anderen Daten bleiben unverändert, so daß die vollständige Übersetzung eines Satzes nicht mehr möglich ist, wenn keine geeigneten Schnittstellen von der Anapherinterpretation zur konzeptuellen Analyse existieren.

Anschließend wird abgefragt, wie der Parameter `default` (siehe Seite 107) für den Ladevorgang gesetzt werden soll. Wird die Frage mit ☐ `yes` ☐ `Return` beantwortet, so wird dieser Parameter auf `on` gesetzt und die Dateien, die in der Datei `mtmenu.pl` spezifiziert sind (siehe Abschnitt 3.5.1 und Anhang B), werden ohne weitere Interaktion mit dem Benutzer geladen, vorausgesetzt, die Dateien existieren. Wird die Frage mit ☐ `no` ☐ `Return` beantwortet, so wird der Parameter `default` auf `off` gesetzt und die linguistischen Daten für die Anapherninterpretation werden interaktiv abgefragt. Der Parameter `default` behält nach dem Laden den in der Laderoutine angegebenen Wert. Anschließend werden die linguistischen Daten für die Anapherninterpretation in der folgenden Reihenfolge geladen:

1. Pronomentabelle und
2. Parameter für die Anapherninterpretation.

3.5.7 Laden der Daten für die Textrepräsentation

☐ `load_text_representation +Sprachbezeichner` ☐ `Return`

Synonyme: `ltr/1`

1. *Sprachbezeichner* ist ein Atom (das erste Argument der Fakten mit den Prädikaten `language_directory/2` und `gram_file/5`, siehe Seite 98).

Mit dem Prädikat `load_text_representation/1` werden die Dateien mit den TBox-Definitionen, die für die Textrepräsentation in der ABox benötigt werden, neu in das MÜ-System geladen. Alle anderen Daten bleiben unverändert, so daß die vollständige Übersetzung eines Satzes nicht mehr möglich ist, wenn keine geeigneten Schnittstellen von der ABox zur konzeptuellen Analyse und zur Anapherninterpretation existieren.

Anschließend wird abgefragt, wie der Parameter `default` (siehe Seite 107) für den Ladevorgang gesetzt werden soll. Wird die Frage mit ☐ `yes` ☐ `Return` beantwortet, so wird dieser Parameter auf `on` gesetzt und die Dateien, die in der Datei `mtmenu.pl` spezifiziert sind (siehe Abschnitt 3.5.1 und Anhang B), werden ohne weitere Interaktion mit dem Benutzer geladen, außer wenn eine der Dateien nicht existiert. Wird die Frage mit ☐ `no` ☐ `Return` beantwortet, so wird der Parameter `default` auf `off` gesetzt und die Dateien mit dem terminologischen Wissen für die Textrepräsentation werden interaktiv abgefragt. Der Parameter `default` behält nach dem Laden den in der Laderoutine angegebenen Wert. Anschließend wird das BACK-System initialisiert und danach eine Liste von Dateien mit TBox-Definitionen geladen.

3.5.8 Laden der Daten für den Transfer

☐ `load_transfer +AS-Sprachbezeichner --> +ZS-Sprachbezeichner` ☐ `Return`

Synonyme: `lt/1`

1. *AS-Sprachbezeichner* ist ein Atom (Ausgangssprache, das erste Argument der Fak-

ten mit den Prädikaten `language_directory/2` und `gram_file/5`, siehe Seite 98).

2. *ZS-Sprachbezeichner* ist ein Atom (Zielsprache, das erste Argument der Fakten mit den Prädikaten `language_directory/2` und `gram_file/5`, siehe Seite 98).

Mit dem Prädikat `load_transfer/1` werden die linguistischen Daten, die für den Transfer benötigt werden, neu in das MÜ-System geladen. Alle anderen Daten bleiben unverändert, so daß die vollständige Übersetzung eines Satzes nicht mehr möglich ist, wenn keine geeigneten Schnittstellen vom Transfer zur semantischen Analyse und zur Generierung existieren.

Anschließend wird abgefragt, wie der Parameter `default` (siehe Seite 107) für den Ladevorgang gesetzt werden soll. Wird die Frage mit ☐ `yes` ☐ `Return` beantwortet, so wird dieser Parameter auf `on` gesetzt und die Dateien, die in der Datei `mtmenu.pl` spezifiziert sind (siehe Abschnitt 3.5.1 und Anhang B), werden ohne weitere Interaktion mit dem Benutzer geladen, es sei denn, daß eine der Dateien nicht existiert. Wird die Frage mit ☐ `no` ☐ `Return` beantwortet, so wird der Parameter `default` auf `off` gesetzt und die linguistischen Daten für den Transfer werden interaktiv abgefragt. Der Parameter `default` behält nach dem Laden den in der Laderoutine angegebenen Wert. Anschließend werden die linguistischen Daten für den Transfer in der folgenden Reihenfolge geladen:

1. Spezifikation für ausgangssprachliche FAS-Kategorienterme,
2. Spezifikation für zielsprachliche FAS-Kategorienterme,
3. FAS-Grammatik der Zielsprache,
4. Abbildungslexikon $FAS \rightarrow TBox$ der Zielsprache¹³ und
5. Termersetzungssystem für die Abbildung von ausgangs- in zielsprachliche FAS-Ausdrücke.

3.5.9 Laden der Daten für die Generierung

☐ `load_generator +Sprachbezeichner` ☐ `Return`

Synonyme: `lg/1`

1. *Sprachbezeichner* ist ein Atom (das erste Argument der Fakten mit den Prädikaten `language_directory/2` und `gram_file/5`, siehe Seite 98).

Mit dem Prädikat `load_generator/1` werden die linguistischen Daten, die für die Generierung benötigt werden, neu in das MÜ-System geladen. Alle anderen Daten bleiben unverändert, so daß die vollständige Übersetzung eines Satzes nicht mehr möglich ist, wenn keine geeigneten Schnittstellen von der Generierung zum Transfer existieren.

Anschließend wird abgefragt, wie der Parameter `default` (siehe Seite 107) für den Ladevorgang gesetzt werden soll. Wird die Frage mit ☐ `yes` ☐ `Return` beantwortet, so wird dieser Parameter auf `on` gesetzt und die Dateien, die in der Datei `mtmenu.pl` spezifiziert sind (siehe Abschnitt 3.5.1 und Anhang B), werden ohne weitere Interaktion mit dem Benutzer geladen, es sei denn, daß es eine der Dateien nicht gibt. Wird die Frage mit

¹³Der TBox-Anteil ist dabei unspezifiziert, d.h. eine anonyme Variable.

no **Return** beantwortet, so wird der Parameter **default** auf **off** gesetzt und die linguistischen Daten für die Generierung werden interaktiv abgefragt. Der Parameter **default** behält nach dem Laden den in der Laderoutine angegebenen Wert. Anschließend werden die linguistischen Daten für die Generierung in der folgenden Reihenfolge geladen:

1. Spezifikation für FAS-Kategorienterme,
2. Spezifikation für hybride Kategorienterme,¹⁴
3. Spezifikation für syntaktische Kategorienterme,
4. kontextfreie Grammatik für die Zielsprache,
5. Lexikon für die Zielsprache,
6. Kasus-Rollen-Tabellen,
7. Termersetzungssystem für die Abbildung von FAS-Ausdrücken in syntaktische Strukturen mit komplexen Kategorien,
8. Abbildungslexikon $FAS \rightarrow TBox$ der Zielsprache,¹⁵
9. morphologisches Lexikon und
10. Flexionsparadigmen.

3.6 Veränderung von MÜ-System-Parametern

switch *+Parameter* *+Wert* **Return**

Synonyme: **sw**/ $\{1,2\}$

1. *Parameter* ist ein Atom (MÜ-System-Parameter).
2. *Wert* ist ein Atom **on** oder **off**. Das Argument ist optional und wird interaktiv abgefragt, falls es nicht angegeben wird.

Mit dem Prädikat **switch**/ $\{1,2\}$ können die Parameter für das MÜ-System, d.h. allgemeine Parameter, für den GPSG-Parser und für das Termersetzungsverfahren verändert werden. Wird bei der interaktiven Eingabe des Wertes nur **Return** gedrückt, so bleibt der System-Parameter unverändert. Nach der Eingabe des Befehls wird der neue Wert ausgegeben. Falls der angegebene Parameter nicht existiert oder der Wert nicht möglich ist, wird eine entsprechende Meldung ausgegeben. Die Parameter, die im MÜ-System gesetzt werden können, sind in den Abschnitten 3.6.1 bis 3.6.3 beschrieben.

switches **Return**

Synonyme: keine definiert

Mit dem Prädikat **switches**/0 können die aktuellen Werte sämtlicher Parameter des MÜ-Systems (siehe Abschnitte 3.6.1 bis 3.6.3) abgefragt werden. Nach der Eingabe des Befehls wird eine Liste mit den System-Parametern und ihren Werten ausgegeben. Die Parameter des MÜ-Systems sind mit den im folgenden Beispiellauf angegebenen Werten initialisiert.

¹⁴Hybride Kategorien enthalten sowohl syntaktische als auch semantische Informationen.

¹⁵Der TBox-Anteil ist dabei unspezifiziert, d.h. eine anonyme Variable.

Beispielanfrage mit switches Return

```
MT(g-->e)> switches
```

```
>>> Parameter "display" is "on".
>>> Parameter "extend" is "off".
>>> Parameter "default" is "on".
>>> Parameter "time" is "off".

>>> Parameter "fip" is "off".
>>> Parameter "alias" is "on".
>>> Parameter "idtrace" is "off".
>>> Parameter "constr" is "off".
>>> Parameter "parser" is "off".
>>> Parameter "edge" is "off".
>>> Parameter "propose" is "off".
>>> Parameter "count" is "off".
>>> Parameter "ap" is "off".
>>> Parameter "hfc" is "off".

>>> Parameter "syntax" is "on".

>>> ok
```

3.6.1 Allgemeine Parameter

Parameter display

Zweck: Ausgabe der Ein- und Ausgabestrukturen (Ableitungsbäume) des GPSG-Parsers und des Termersetzungsverfahrens auf dem Bildschirm

Wert = on: Sämtliche Strukturen werden ausgegeben. Ausgenommen davon ist die Komponente zur Übersetzung eines Satzes von der Ausgangs- in die Zielsprache mit dem Befehl `translate/0` (siehe Abschnitt 3.3.9), in der unabhängig vom Wert des Parameters keine Strukturen ausgegeben werden.

Wert = off: Anstatt der Strukturen werden nur die entsprechenden Ketten von terminalen Symbolen ausgegeben.

Initialer Wert: on

Parameter extend

Zweck: Ausgabe der komplexen Kategorien, die in den Eingabe-, Zwischen- und Ausgabestrukturen der morphologischen Analyse, des Parsers und des TE-Verfahrens vorkommen. Die Zwischenstrukturen werden nur beim Aufruf der Trace-Routinen (siehe Abschnitt 3.3.1) für das TE-Verfahren ausgegeben.

Wert = on: Die komplexen Kategorien werden in erweiterter Notation ausgegeben, d.h. es werden nicht nur die Merkmalwerte als Argumente der Hauptkategorie (eines Prädikates) ausgegeben (z.B. `dp(gpsg,nom,top,^,^)`), sondern auch die entsprechenden Merkmalnamen mit Doppelpunkt vorangestellt (z.B. `dp(lang:gpsg,cas:nom,them:top,refl:^,slash1:^)`). In der Regel sind die Strukturen dadurch größer, so daß dabei der Überblick über eine Struktur verlorengehen kann.

Diese Ausgabe benötigt für die Ausgabe von Strukturen während des TE-Verfahrens die Dateien mit den Definitionen der komplexen Kategorien der beteiligten Strukturen (GPSG, hybrid, FAS oder ATL), die mit den Kürzeln **gdef**, **hdef**, **fdef** bzw. **adef** in der Datei **mtmenu.pl** (siehe Abschnitt 3.5.1 und Anhang B) spezifiziert sind. Wenn diese Dateien nicht geladen wurden oder zu einer Hauptkategorie die entsprechenden Definitionen nicht in der Datenbasis vorhanden sind, dann werden die komplexen Kategorien in einfacher Form ausgegeben (als hätte der Parameter den Wert **off**).

Wert = off: Die komplexen Kategorien werden in normaler Form ausgegeben, d.h. als Argumente einer Hauptkategorie (eines Prädikats) werden nur die Werte der für sie definierten Merkmale ausgegeben (z.B. **dp(gpsg,nom,top,^,^)**). Dies ist kürzer und verbessert den Gesamtüberblick über eine Struktur.

Initialer Wert: off

Parameter default

Zweck: Laden von linguistischen Daten für das MÜ-System oder für einzelne Komponenten (siehe Abschnitt 3.5). Vor Beginn jeder Laderoutine wird interaktiv abgefragt, auf welchen Wert der Parameter **default** (siehe Seite 107) gesetzt werden soll.

Wert = on: Die in der Datei **mtmenu.pl** (siehe Abschnitt 3.5.1 und Anhang B) spezifizierten Dateien (Defaultdateien) werden durch die Laderoutinen geladen.

Wert = off: Die Dateinamen werden durch die Laderoutinen interaktiv erfragt.

Initialer Wert: on

Parameter time

Zweck: Ausgabe der realen Verarbeitungszeit von einzelnen Komponenten des MÜ-Systems in Sekunden

Wert = on: Jede Komponente gibt die reale Verarbeitungszeit in Sekunden aus.

Wert = off: Es erfolgen keine zusätzlichen Ausgaben über die Verarbeitungszeit.

Initialer Wert: off

3.6.2 Parameter für den Parser

Parameter fip (*feature instantiation principles*)

Zweck: Ausgabe von Informationen über die Anwendung der Merkmalinstantiierungsprinzipien (FIPs) auf lokale GPSG-Strukturen und der Feature-Co-occurrence-Restrictions (FCRs) auf GPSG-Kategorien. Das Foot-Feature-Prinzip (FFP) wird in jedem Fall angewendet. Die Anwendung der Head-Feature-Convention (HFC) und des Agreement-Prinzips (AP) werden durch den System-Parameter **hfc** bzw. **ap** (siehe Seite 114) gesteuert.

Wert = on: Die lokalen GPSG-Strukturen werden vor und nach der Anwendung der FIPs in indentierter Baumnotation ausgegeben (siehe Abschnitt 3.2.5). Außerdem werden die Nummer und die Kurzform (Kommentar, siehe Seite 59) der ID-Regel ausgegeben, welche die lokale Struktur lizenziert.

Falls der Parameter **alias** (siehe Seite 108) den Wert **on** hat, werden sämtliche komplexe Kategorien der lokalen GPSG-Struktur in Kurzform ausgegeben.

Falls der Parameter **extend** (siehe Seite 106) den Wert **on** hat, werden sämtliche Kategorien der lokalen GPSG-Struktur in erweiterter Notation ausgegeben.

Die Ausgaberroutine benötigt die Datei mit den Kurzformen (Kommentaren) der ID-Regeln, die unter dem Kürzel **idc** in der Datei **mtmenu.pl** (siehe Abschnitt 3.5.1 und Anhang B) angegeben ist. Wenn diese Datei nicht geladen wurde oder zu einer ID-Regel kein Kommentar vorhanden ist, dann wird nur deren Nummer genannt.

Wert = off: Es erfolgen keine zusätzlichen Ausgaben über die Anwendung der FIPs.

Initialer Wert: off

Beispiel für die Ausgabe bei fip = on

```
>>> #####
>>> FIPs: ID(38) DP --> D1, (V3rel)
>>> #####
>>>         dp(3,+,acc,THEM0)
>>>         --- d1(3,+,acc)rel[3,+,GEN0,r]
>>>             ==FIP==>
>>>         dp(3,+,acc,THEM1)rel[3,+,GEN0,r]
>>>         --- d1(3,+,acc)rel[3,+,GEN0,r]
>>>
>>> #####
>>> FIPs: ID(39) DP --> D1
>>> #####
>>>         dp(3,+,acc,THEM2)// (v3rel(PER1,PLU1,foc,AUX1,PAST1)rel[3,+,GEN0,r])
>>>         --- d1(3,+,acc)rel[3,+,GEN0,r]
>>>             ==FIP==>
>>>         FFP failed !
>>>         local tree not admissible!
```

Parameter alias

Zweck: Ausgabe von komplexen GPSG-Kategorien in abgekürzter Form

Wert = on: Sämtliche GPSG-Kategorien bei der morphologischen Analyse, in den GPSG-Strukturen, bei zusätzlichen Ausgaben werden in abgekürzter Form ausgegeben. Beispielsweise wird statt der internen GPSG-Kategorie `cat(n,0,1,3,-,2,CAS,FLEX, ^,Q,^,^,^, ^,^,^,^, CO,CONJ,^, ^,^,^,^, ^,^,^,^, AGR,HEAD, ^,^,^, ^)` die abgekürzte Form `n(1,-,2,CAS)` ausgegeben, in der nur die Werte der für sie relevanten Merkmale vorkommen, d.h., daß in der Regel die Werte '^' (undefiniert) nicht mit ausgegeben werden.

Wert = off: Sämtliche GPSG-Kategorien werden in ihrem internen Format mit den Werten aller für sie definierten Merkmale ausgegeben.

Initialer Wert: on

Parameter idtrace

Zweck: Ausgabe über die Zulässigkeit von lokalen GPSG-Strukturen bezüglich der LP-Aussagen, der Merkmalinstantiierungsprinzipien (FIPs) und der Feature-Co-occurrence-Restrictions (FCRs). Von den FIPs wird das Foot-Feature-Prinzip (FFP) in jedem Fall angewendet. Die Anwendung der Head-Feature-Convention (HFC) und des Agreement-Prinzips (AP) werden durch den System-Parameter `hfc` bzw. `ap` (siehe Seite 114) gesteuert.

Wert = on: Es erfolgen Meldungen vor und nach der Überprüfung der LP-Konsistenz, der FIP-Zulässigkeit von lokalen GPSG-Strukturen, die von einer ID-Regel lizenziert werden, und der Legalität von GPSG-Kategorien gemäß den FCRs.

Der Aufruf jeder der drei Komponenten LP, FIP oder FCR wird gemeldet. Nach Abschluß wird berichtet, ob die Anwendung erfolgreich war oder nicht. Widerspricht die lokale Struktur einem der angewendeten FIPs (FFP, evtl. HFC oder AP), so kann nur festgestellt werden, welches der FIPs fehlschlug, indem der Parameter `fip` auf `on` gesetzt (siehe Seite 107) und anschließend die Analyse wiederholt wird.

Die Ausgaberoutine benötigt die Datei mit den Kurzformen (Kommentaren) der ID-Regeln, die unter dem Kürzel `idc` in der Datei `mtmenu.pl` (siehe Abschnitt 3.5.1 und Anhang B) angegeben ist. Wenn diese Datei nicht geladen wurde oder zu einer ID-Regel kein Kommentar vorhanden ist, dann wird nur deren Nummer ausgegeben.

Wert = off: Es erfolgen keine zusätzlichen Ausgaben über den Erfolg bzw. Mißerfolg bei der Anwendung der LP-Aussagen, FIPs oder FCRs auf lokale GPSG-Strukturen bzw. -Kategorien.

Initialer Wert: off

Parameter constr

Zweck: Die Anwendung der LP-Aussagen und der Feature-Co-occurrence-Restrictions (FCRs) der GPSG ist immer dann problematisch, wenn die beteiligten Kategorien in einer lokalen Struktur zwar mit den entsprechenden Kategorien in den LP-Aussagen bzw. den Bedingungskategorien der FCRs unifizierbar, aber keine Extensionen sind. Die Einschränkungen für die Zulässigkeit von lokalen Bäumen gemäß den LP-Aussagen und den FCRs (Legalität der komplexen Kategorien) werden als prädikatenlogische Aussage in konjunktiver Normalform (KNF) repräsentiert und ausgewertet (siehe Abschnitt 1.2.2).

Wert = on: Im Falle des Widerspruchs in einer Disjunktion von Constraints in einer konjunktiven Normalform, die aus einer LP-Aussage oder FCR ermittelt wurde, wird eine entsprechende Meldung ausgegeben.

Wert = off: Es erfolgen keine zusätzlichen Ausgaben über die Ablehnung von GPSG-Strukturen durch LP- oder FCR-Constraints.

Initialer Wert: off

Parameter parser

Zweck: Ausgabe über die laufenden Aktionen des Parsers

Wert = on: Es erfolgen zusätzliche Ausgaben über die Aktionen des Parsers, d.h. wenn eine Wortform gelesen oder versucht wurde, eine lokale GPSG-Struktur zu reduzieren. Die Erkennung einer Wortform wird angezeigt, indem die Wortform, die prä-terminale GPSG-Kategorie und der Wert für das Merkmal Tempus ausgegeben wird. Die Ausgabe des Tempus erfolgt separat, da diese Information für die Syntax nicht relevant und deshalb nicht als Merkmal in der GPSG definiert ist. Bei nicht verbalen Kategorien wird **undef** (Tempus undefiniert) ausgegeben.

Wenn der Parser versucht, eine lokale GPSG-Struktur zu konstruieren, die von einer ID-Regel lizenziert wird, erfolgt die Ausgabe der lokalen Struktur und der ID-Regel in Kurzform (siehe Seite 59). War die Reduktion der lokalen Struktur durch die ID-Regel möglich, so wird eine entsprechende Meldung ausgegeben.

Falls der Parameter **alias** (siehe Seite 108) den Wert **on** hat, werden sämtliche GPSG-Kategorien des lokalen Baumes in Kurzform ausgegeben.

Falls der Parameter **extend** (siehe Seite 106) den Wert **on** hat, werden sämtliche GPSG-Kategorien des lokalen Baumes in erweiterter Notation ausgegeben.

Die Ausgaberroutine benötigt die Datei mit den Kurzformen (Kommentaren) der ID-Regeln, die unter dem Kürzel **idc** in der Datei **mtmenu.pl** (siehe Abschnitt 3.5.1 und Anhang B) angegeben ist. Wenn diese Datei nicht geladen wurde oder zu einer ID-Regel kein Kommentar vorhanden ist, dann wird nur deren Nummer ausgegeben.

Wert = off: Es erfolgen keine zusätzlichen Ausgaben über die Aktionen des Parsers.

Initialer Wert: off

Beispiel für die Ausgabe bei `parser = on`

```
>>> read: die --- reduced to d(1,3,-,2,acc,1)rel[3,-,2,r] undef
>>> =====
>>> Try to reduce ID(41) "D1 --> d[1]"
>>>         d1(3,-,acc)
>>>         --- d(1,3,-,2,acc,1)rel[3,-,2,r]
>>>
>>> ID(41) reduced to d1(3,-,acc)rel[3,-,2,r]
>>> =====
>>> =====
>>> Try to reduce ID(38) "DP --> D1, (V3rel)"
>>>         dp(3,-,acc,THEM0)
>>>         --- d1(3,-,acc)rel[3,-,2,r]
>>>
>>> ID(38) reduced to dp(3,-,acc,THEM1)rel[3,-,2,r]
>>> =====
>>> =====
>>> Try to reduce ID(39) "DP --> D1"
>>>         dp(3,-,acc,THEM2)// (v3rel(PER1,PLU1,foc,AUX1,PAST1)rel[3,-,2,r])
>>>         --- d1(3,-,acc)rel[3,-,2,r]
>>>
>>> read: die --- reduced to d(1,3,+,GEN0,nom,1)rel[3,+,GEN0,r] undef
>>> =====
```

```

>>> Try to reduce ID(41) "D1 --> d[1]"
>>>         d1(3,+,nom)
>>>         --- d(1,3,+,GEN0,nom,1)rel[3,+,GEN0,r]
>>>
>>> ID(41) reduced to d1(3,+,nom)rel[3,+,GEN0,r]
>>> =====
>>> =====
>>> Try to reduce ID(38) "DP --> D1, (V3rel)"
>>>         dp(3,+,nom,THEM0)
>>>         --- d1(3,+,nom)rel[3,+,GEN0,r]
>>>
>>> ID(38) reduced to dp(3,+,nom,THEM1)rel[3,+,GEN0,r]
>>> =====

```

Parameter edge

Zweck: Ausgabe der Kanten, die der Parser in die Chart einfügt

Wert = on: Immer wenn der Parser (siehe Abschnitt 1.2.2) eine Kante in die Chart einfügt, werden ihre GPSG-Kategorie, ihr Anfang, ihr Ende und die Dominanzmenge der GPSG-Kategorie ausgegeben.

Mit der GPSG-Kategorie ist die Kante der Chart etikettiert. Falls der Parameter **alias** (siehe Seite 108) den Wert **on** hat, wird die Kategorie in Kurzform ausgegeben.

Falls der Parameter **extend** (siehe Seite 106) den Wert **on** hat, wird die Kategorie in erweiterter Notation ausgegeben.

Der Anfang und das Ende einer Kante sind natürliche Zahlen. Dabei kann man sich die Zahlen so vorstellen, daß sie die Wortzwischenräume bezeichnen, d.h. die Ausgabe $\langle \dots, 2, 4, \dots \rangle$ bedeutet, daß im eingegebenen Satz eine Konstituente erkannt wurde, die das dritte und vierte Wort des Satzes umfaßt.

Die Dominanzmenge ist eine Prolog-Liste mit Nummern von ID-Regeln, in denen die entsprechende GPSG-Kategorie als Tochter zugelassen ist.

Wert = off: Es erfolgen keine zusätzlichen Ausgaben über die Kanten der Chart.

Initialer Wert: off

Beispiel für die Ausgabe bei edge = on

```

>>> <d(1,3,+,GEN0,acc,1)rel[3,+,GEN0,r],0,1,[41]>
>>> <d1(3,+,acc)rel[3,+,GEN0,r],0,1,[38,39]>
>>> <dp(3,+,acc,THEM1)rel[3,+,GEN0,r],0,1,[8,12,14,20,21,24,29,31,55,65]>
>>> <d(2,3,-,2,nom,1),0,1,[43]>
>>> <d(2,3,-,2,acc,1),0,1,[43]>
>>> <d(2,3,+,GEN0,nom,1),0,1,[43]>
>>> <d(2,3,+,GEN0,acc,1),0,1,[43]>
>>> <a(1,3,-,2,gen,1),1,2,[58]>
>>> <ap(-,gen),1,2,[29,30,45,70]>
>>> <a(1,3,-,2,gen,2),1,2,[58]>
>>> <ap(-,gen),1,2,[29,30,45,70]>
>>> <a(1,3,-,2,dat,1),1,2,[58]>
>>> <ap(-,dat),1,2,[29,30,45,70]>

```

Parameter propose

Zweck: Vorschläge des Parsers für unbekannte Wortformen

Wert = on: Unbekannte Wortformen, die eingegeben wurden, werden schon während der morphologischen Analyse erkannt. In diesem Fall muß ein anderes Wort eingegeben werden. Dies wiederholt sich solange, bis die morphologische Analyse erfolgreich ist. Der Parser wird erst dann aufgerufen, wenn sämtliche eingegebenen Wortformen morphologisch analysiert werden konnten, so daß er keine Vorschläge für unbekannte Wortformen machen muß.

Wert = off: Wenn eine eingegebene Wortform nicht morphologisch analysiert werden kann, wird diese Komponente übersprungen. Der Parser versucht dann, für diese Wortform eine "passende" Kategorie zu finden, so daß die Analyse erfolgreich abgeschlossen werden kann, wenn möglich. In der syntaktischen Struktur wird die unbekannte Wortform mit dem Suffix '?' versehen. Dies führt allerdings in der Regel zu erheblich längeren Analysezeiten.

Initialer Wert: off

Beispiel für die Ausgabe bei propose = on

```
MT(g-->e)> switch propose on
>>> Parameter "propose" was set to "on" !

>>> ok

MT(g-->e)> parse
>>> Please input a sentence ending with "[RETURN]" !

Input> er xyz ihn

>>> Lemmatization for "er".
d(1,3,-,1,nom,FLEX0)
-----
>>> Lemmatization for "xyz".
>>> proposing
-----
>>> Lemmatization for "ihn".
d(1,3,-,1,acc,FLEX0)
-----

>>> Input: er xyz ihn

>>> dominance chart parser running ...

v3(3,-,THEM0,-,+,+,fin,PAST0)
--- dp(3,-,nom,top)
----- d1(3,-,nom)
----- d(1,3,-,1,nom,FLEX1) er
--- v3(3,-,^,-,+,+,fin,PAST0)/dp(3,-,nom,top)
----- v(6,3,-,^,-,PREFIX6,+,+,fin,PAST0) xyz ?
----- dp(3,-,acc,THEM7)
----- d1(3,-,acc)
----- d(1,3,-,1,acc,FLEX7) ihn
```

```

>>> Result stored with label "gpsg8" under predicate "gpsg/2" !

v3(3,-,THEM0,-,+,+,fin,PAST0)
--- dp(3,-,nom,top)
----- d1(3,-,nom)
----- d(1,3,-,1,nom,FLEX1) er
--- v3(3,-,^,-,+,+,fin,PAST0)/dp(3,-,nom,top)
----- v(31,3,-,^,-,PFIX6,+,+,fin,PAST0) xyz ?
----- dp(3,-,acc,THEM7)
----- d1(3,-,acc)
----- d(1,3,-,1,acc,FLEX7) ihn

>>> Result stored with label "gpsg8" under predicate "gpsg/2" !

>>> Analysis terminates successfully !

>>> ok

```

Parameter count

Zweck: Ausgabe der Anzahl der Kanten der Chart und sämtlicher syntaktischen Strukturen, welche alle eingegebenen Wortformen umfassen

Wert = on: Unabhängig von einer erfolgreichen syntaktischen Analyse wird nach deren Abschluß die Gesamtanzahl der Kanten ausgegeben, die im Laufe der Analyse in die Chart eingefügt wurden.

Außerdem werden sämtliche Ergebnisstrukturen ausgegeben, die alle eingegebenen Wortformen umfassen, d.h. es werden auch die GPSG-Strukturen ausgegeben, in deren Wurzelkategorie ein kategorienwertiger Merkmalwert vorkommt.

Wert = off: Es erfolgt keine zusätzliche Ausgabe über die Anzahl der Kanten, die während der syntaktischen Analyse in die Chart eingefügt wurden.

Außerdem werden nur die Ergebnisstrukturen der syntaktischen Analyse ausgegeben, die alle eingegebenen Wortformen umfassen und in deren Wurzel kein kategorienwertiger Merkmalwert vorkommt.

Initialer Wert: off

Parameter ap (*agreement principle*)

Zweck: Anwendung des Agreement-Prinzips (AP) auf lokale GPSG-Strukturen

Wert = on: Das Agreement-Prinzip (AP, siehe Abschnitt 1.1.2) wird auf sämtliche lokalen GPSG-Strukturen angewendet. Bei allen vordefinierten Grammatiken wurde das AP bereits in einem Vorverarbeitungsschritt angewendet. Aus diesem Grund ist der Wert **on** nur sinnvoll, wenn ID-Regeln neu definiert werden, auf die das AP nicht vorher angewendet wird.

Wert = off: Das AP wird auf keine lokale GPSG-Struktur angewendet. Dies ist sinnvoll, wenn das AP vorher auf die ID-Regeln angewendet wurde. Bei allen vordefinierten Grammatiken ist dies der Fall.

Initialer Wert: off

Parameter <code>hfc</code> (<i>head feature convention</i>)
--

Zweck: Anwendung der Head-Feature-Convention (HFC) auf lokale GPSG-Strukturen

Wert = on: Die Head-Feature-Convention (HFC, siehe Abschnitt 1.1.2) wird auf sämtliche lokale GPSG-Strukturen angewendet. Bei allen vordefinierten Grammatiken wurde die HFC bereits in einem Vorverarbeitungsschritt angewendet. Aus diesem Grund ist der Wert **on** nur sinnvoll, wenn ID-Regeln neu definiert werden, auf welche die HFC nicht vorher angewendet wird.

Wert = off: Die HFC wird auf keine lokale GPSG-Struktur angewendet. Dies ist sinnvoll, wenn die HFC vorher auf die ID-Regeln angewendet wurde. Bei allen vordefinierten Grammatiken ist dies der Fall.

Initialer Wert: off

3.6.3 Parameter für das TE-Verfahren

Parameter <code>syntax</code>

Zweck: Überprüfung der Korrektheit einer irreduziblen Struktur gemäß der Zielsyntax. Ein Term (eine Struktur) heißt irreduzibel oder minimal, wenn auf ihn keine TE-Regel mehr anwendbar ist.

Wert = on: Irreduzible Strukturen werden auf syntaktische Korrektheit überprüft, so daß nur Ergebnisse erzeugt werden, die der jeweiligen Zielsyntax entsprechen.

Die Überprüfung der Korrektheit erfordert die Dateien mit den kontextfreien Grammatiken der jeweiligen Zielstruktur. Diese Dateien sind unter den Kürzeln **fas**, **atlgam**, bzw. **idgen** in der Datei **mtmenu.pl** (siehe Abschnitt 3.5.1 und Anhang B) angegeben. Wenn diese Dateien nicht geladen wurden, so gibt es in jedem Fall kein Ergebnis des TE-Verfahrens für eine Eingabestruktur.

Wert = off: Sämtliche irreduzible Strukturen werden als Ergebnis des TE-Verfahrens erzeugt. Sie werden vorher nicht auf syntaktische Korrektheit überprüft.

Dies ist dann sinnvoll, wenn das TE-Verfahren für eine Eingabestruktur keine syntaktisch korrekte Zielstruktur ableiten kann. In diesem Fall wird der Parameter **Syntax** auf **off** gesetzt und dieselbe Struktur nochmals als Eingabe für das TE-Verfahren verwendet, so daß nun auch alle irreduziblen, nicht syntaktischen Strukturen ausgegeben werden. An den ausgegebenen Strukturen läßt sich in der Regel leicht erkennen, wo der Fehler lag (oft ist es eine fehlende TE-Regel oder die linke Seite einer vorhandenen TE-Regel ist falsch definiert).

Initialer Wert: on

3.6.4 Parameter für das Anapherninterpretationsverfahren

Die Anapherninterpretationskomponente des MÜ-Systems (siehe Abschnitt 1.2.4) verwendet neben einer referentiellen Textrepräsentation eine strukturelle Textrepräsentation (siehe Abschnitt 1), um einen Antezedenskandidaten nach strukturellen Kriterien zu bewerten. Die strukturelle Textrepräsentation wird im MÜ-System als Liste von FAS-Ausdrücken realisiert, die als Vortext bezeichnet wird. Die Liste enthält Label für bereits

ausgewertete FAS-Ausdrücke. Falls vom Benutzer nicht anders definiert, enthält sie die Label für die ausgewerteten FAS-Ausdrücke der Sätze, die dem aktuellen Satz in einem Text vorausgehen, d.h. vom MÜ-System vor dem aktuellen Satz verarbeitet und abgespeichert wurden. Die Länge des Vortextes beträgt initial 1, sie kann aber durch den Benutzer verändert werden. Die Vortextliste wird vom MÜ-System nach der Verarbeitung eines Satzes aktualisiert. Die Liste ist geordnet. Je aktueller der Satz, desto weiter vorne in der Liste kommt das entsprechende Label vor, d.h. das erste Element repräsentiert den zuletzt verarbeiteten Satz, das zweite Element den vorletzten Satz usw.

Ein Vortext existiert sowohl für die Ausgangs- als auch für die Zielsprache. Für beide Sprachen ist er gleich lang. Allerdings wird das Anapherninterpretationsverfahren nur auf die Ausgangssprache angewendet. Das Verfahren betrachtet zunächst sämtliche Nominalphrasen aus dem Vortext als mögliche Antezedens Kandidaten, darunter auch Pronomina und koordinierte Nominalphrasen, und setzt voraus, daß in der Vortextliste Label **fasn** von ausgewerteten FAS-Ausdrücken vorkommen, die unter dem Prädikat **eval_fas/2** nach der konzeptuellen Analyse in der Datenbasis gespeichert wurden (siehe Abschnitt 3.3.5).

Auf der zielsprachlichen Seite werden die anaphorischen Bezüge der Ausgangssprache durch den Transfer einfach übernommen. Das ist zwar nicht adäquat, aber als Behelfslösung einigermaßen akzeptabel. In einem zielsprachlichen FAS-Ausdruck werden die Kongruenzmerkmale eines Pronomens mit denen seines Antezedenten, der in der Ausgangssprache ermittelt wurde, gleichgesetzt. Hier wird vorausgesetzt, daß in der Vortextliste Label **genn** von FAS-Ausdrücken vorkommen, die unter dem Prädikat **gen/2** nach dem Transfer in der Datenbasis abgespeichert werden (siehe Abschnitt 3.3.6).

init_vortext <i>+Bezeichner</i>	Return
--	---------------

Synonyme: **iv/1**

1. *Bezeichner* ist ein Atom **source** oder **target** (Ausgangs- bzw. Zielsprache).

Mit dem Prädikat **init_vortext/1** wird der Vortext für die Ausgangs- bzw. Zielsprache initialisiert, d.h. die entsprechende Liste auf die leere Liste gesetzt. Die maximale Länge bleibt dabei unverändert.

change_vortext <i>+Bezeichner +Liste</i>	Return
---	---------------

Synonyme: **cv/2**

1. *Bezeichner* ist ein Atom **source** oder **target** (Ausgangs- bzw. Zielsprache).
2. *Liste* ist eine Liste mit Labeln **eval_fasn** bzw. **genn** für ausgewertete FAS-Ausdrücke der Ausgangssprache bzw. für FAS-Ausdrücke der Zielsprache.

Mit dem Prädikat **change_vortext/2** wird die Vortextliste für die Ausgangs- bzw. Zielsprache auf *Liste* gesetzt. Die Länge bleibt dabei unverändert.

Wenn *Liste* ein Label für einen FAS-Ausdruck enthält, der nicht unter dem Prädikat **eval_fas/2** (für den Bezeichner **source**) bzw. **gen/2** (für den Bezeichner **target**) in der Datenbasis gespeichert ist, so wird eine entsprechende Fehlermeldung ausgegeben.

<code>set_vortext_length</code> <i>+Länge</i>	Return
---	---------------

Synonyme: sv1/1

1. *Länge* ist eine positive ganze Zahl (Länge des Vortextes für die Ausgangs- und Zielsprache).

Mit dem Prädikat `set_vortext_length/1` wird die maximale Länge des Vortextes für die Ausgangs- und Zielsprache auf *Länge* gesetzt. Dabei wird die Länge des Vortextes beider Sprachen auf diesen Wert gesetzt. Wird eine negative ganze Zahl angegeben, so wird die Länge des Vortextes auf 0 gesetzt.

<code>display_vortext</code> <i>+Bezeichner</i>	Return
---	---------------

Synonyme: dv/1

1. *Bezeichner* ist ein Atom `source` oder `target` (Ausgangs- bzw. Zielsprache).

Mit dem Prädikat `display_vortext/1` werden die aktuelle Vortextliste und die definierte maximale Länge für die Ausgangs- bzw. Zielsprache ausgegeben.

<code>init_text_block</code>	Return
------------------------------	---------------

Synonyme: itb/0

Das Prädikat `init_text_block/0` dient dazu, mit der Verarbeitung eines völlig neuen, vom vorher eingegebenen Text unabhängigen Textes beginnen zu können. Dazu muß nämlich die referentielle Textrepräsentation in der ABox des BACK-Systems (siehe Abschnitt 1), die Vortextlisten der Ausgangs- und Zielsprache und deren maximale Längen neu initialisiert werden. Dabei wird die ABox gelöscht, die Vortextlisten der Ausgangs- und Zielsprache auf die leere Liste und deren maximale Längen auf 1 gesetzt.

A Das Textfragment

Die in diesem Abschnitt des Anhangs angegebenen Beispielsätze befinden sich in der Datei `frag92`.¹⁶ Die GPSG-Strukturen des Deutschen für die einzelnen Sätze (Ausgangspunkt für die semantische Analyse) stehen in der Datei `gpsg92`, die FAS-Ausdrücke des Deutschen (Ausgangspunkt für die konzeptuelle Analyse) in `fas92`, die ausgewerteten FAS-Ausdrücke des Deutschen (Ausgangspunkt für den Transfer) in `eval_fas92` und die FAS-Ausdrücke des Englischen (Ausgangspunkt für die Generierung) in `gen92`.

Hinter den einzelnen Sätzen des Deutschen sind drei Listen von Zahlen in Klammern angegeben. Die erste Liste enthält die Nummern n der Label `gpsgn` der GPSG-Strukturen des Deutschen in der o.g. Datei. Die zweite Liste enthält die Nummern n der Label `fasn` bzw. `eval_fasn` der (ausgewerteten) FAS-Ausdrücke des Deutschen in den o.g. Dateien. Die Label der nicht ausgewerteten und ausgewerteten FAS-Ausdrücke sind identisch in den beiden Dateien. Die dritte Liste enthält die Nummern n der Label `genn` der FAS-Ausdrücke des Englischen in der o.g. Datei. Das Präfix '+' einer Zahl bedeutet dabei, daß es sich um die intendierte Struktur handelt und das Präfix '-' ist eine nicht intendierte Lesart des Satzes. Zahlen ohne Präfix implizieren zwar mögliche, aber nicht intendierte Strukturen.

D: die usa ergreifen neue initiativen. $(-0, +1)$ (0) (0)

E: the usa are taking new initiatives.

D: sie erhoehen ihre investitionen um ihre technologische industrielle und kommerzielle vorherrschaft zu verbessern. $(-2, +3)$ (1) (1)

E: they increase their investments in order to improve their technological industrial and commercial predominance.

D: die usa ergreifen eine neue initiative um ihre vorherrschaft zu festigen. $(+4)$ (2) (2)

E: the usa are taking a new initiative in order to strengthen their predominance.

D: die vertreter der groessten europaeischen firmen haben sich an die kommission gewandt. $(+5)$ (3) (3)

E: representatives of the largest european companies have approached the commission.

D: sie schrieben an vizepraesident davignon. $(-6, +7)$ (4) (4)

E: they wrote to vice president davignon.

D: das vorliegende dokument stellt die grundlage fuer einen beschluss des rates dar. $(-8, +9, -10, -11)$ (5) (5)

E: the present document provides the basis for a council decision.

D: er soll ein entscheidendes element der industriellen strategie der gemeinschaft sein. $(12, +13)$ (6) (6)

E: it is designed to be an essential element in the industrial strategy of the community.

D: diese strategie wird von der kommission in ihrer mitteilung vorgeschlagen. $(+14)$ (7) (7)

E: this strategy is proposed by the commission in its communication.

¹⁶Sämtliche in diesem Abschnitt des Anhangs genannten Dateien befinden sich in dem Verzeichnis `~/FAST-v94/daten/fragmente`.

D: europaeische firmen werden durch massnahmen ihrer regierungen unterstuetzt. (15, +16) (8) (8)

E: european companies are supported by measures of their governments.

D: die mittel die in der brd aufgewendet werden sind zu gering. (+17) (9) (9)

E: the resources that are committed in the frg are too small.

D: sie sind nicht auf internationalen wettbewerb ausgerichtet. (+18) (10) (10)

E: they are not focussed towards international competition.

D: in ihrer analyse ueber das europaeische forschungspotential hat die gemeinschaft folgende punkte ermittelt. (+19) (11) (11)

E: in its analysis of europe's research potential the commission has identified the following points.

D: die benutzer bekommen zugriff zu allen einschlaegigen informationen die sie zu der loesung ihrer probleme brauchen. (-20, -21, 22, +23, -24, -25, -26, -27, -28, -29) (12) (+12, 13)

E: the users get access to all relevant pieces of information which they need for the solution of their problems.

D: fortgeschrittene systeme erkennen die information in der form in der sie generiert wird. (-30, +31) (13) (14)

E: advanced systems perceive information in the form in which it is generated.

D: sie integrieren sie in das gespeicherte wissen. (-32, +33) (14) (15)

E: they integrate it into the stored knowledge.

D: die informationen sollen vertraulich behandelt werden. (+34) (15) (16, +17, 18)

E: information shall be treated confidentially.

D: der staat der sie liefert fordert dies. (+35) (16) (19)

E: the country that delivers it requests this.

D: die industrie kann daran mitwirken ohne ihre faehigkeit zu dem wettbewerb um maerkte zu beeintraechtigen. (+36, -37, -38) (17) (20)

E: industry can collaborate without impeding its ability to compete for markets.

D: die hochschulen betrachten die erforderlichen arbeiten als allzu anwendungsbezogen. (-39, +40, -41, -42) (18) (21)

E: the universities regard the necessary work as too applied.

D: die industrie sieht sie als reine grundlagenforschung an. (-43, +44, -45, -46) (19) (22)

E: the industry regards it as pure basic research.

D: in den usa haben die groessten hersteller von halbleitern eine initiative ergriffen zu der sie einen grossen beitrug leisten wollen. (+47) (20) (23)

E: in the usa the largest manufacturers of semiconductors have taken an initiative to which they want to contribute a large amount.

- D: diese bereiche wurden angesichts der breite des technologiespektrums ausgewaehlt. (+48) (21) (+24, 25)
 E: these areas were chosen in the light of the width of the technology spectrum.
- D: sie ist fuer ihre entwicklung erforderlich. (-49, -50, +51) (22) (26)
 E: it is neccessary for their development.
- D: neben dieser loesung soll eine infrastruktur geschaffen werden. (+52) (23) (27)
 E: apart from this solution an infrastructure is going to be established.
- D: durch sie koennen systematische informationen zugaenglich gemacht werden. (+53) (24) (28, +29)
 E: by it systematic pieces of information can be made available.
- D: die telekommunikation ist grundlegend fuer informationstechnologien. (-54, +55) (25) (30)
 E: telecommunication is basic for information technologies.
- D: die fuer esprit vorgeschlagenen gebiete werden daher ihre wichtigsten technologischen fragen abdecken. (-56, +57) (26) (31)
 E: * the proposed for esprit areas will cover most of its technological questions.
- D: die organisation der telekommunikationsdienste macht sie zu einer besonderen anwendung. (-58, +59, -60, -61) (27) (32)
 E: the organization of the telecommunication makes it to a special application.
- D: buerosysteme sind ein bedeutender bereich fuer it anwendungen. (+62) (28) (33)
 E: office systems are a significant it application.
- D: ihnen kommt eine grosse bedeutung fuer den handel zu. (+63, -64) (29) (34)
 E: they have a major importance for business.
- D: sie erfordern fortschritte in vielen technologischen bereichen. (-65, +66, -67, -68) (30) (35)
 E: they require advances in many technological areas.
- D: die usa erhoehen ihre initiativen in europa. (-69, +70, -71, 72) (31) (36)
 E: the usa take new initiatives in europe.
- D: sie schreiben an davignon ueber die massnahmen. (-73, +74) (32) (37)
 E: they write to davignon about the measures.
- D: sie integrieren das wissen in der form in der es generiert wird. (+75) (33) (38)
 E: they integrate the knowledge in the form in which it is generated.
- D: die industrie wird durch massnahmen ihrer regierungen unterstuetzt. (76, +77) (34) (39)
 E: the industry is supported by measures of their governments.
- D: er erhoehrt seine investitionen um seine vorherrschaft zu festigen. (+78, -79) (35) (40)
 E: he increases his investments in order to strengthen his predominance.

- D: sie beeinträchtigen ihre fähigkeit zu dem wettbewerb um märkte. (+80, -81, -82) (36) (41)
 E: they influence their ability to the competition for markets.
- D: sie generieren ihre information ueber den fortschritt. (+83, -84) (37) (42)
 E: they generate their information about the advance.
- D: angesichts dieser sache werden die identität europas und seine politische unab-
 hängigkeit gefährdet. (+85) (38) (43)
 E: in these conditions the identity of europe and its political identity will be at stake.
- D: die usa und japan ergreifen neue initiativen. (-86, +87) (39) (44)
 E: the usa and japan are taking new initiatives.
- D: sie erhöhen ihre investitionen um ihre technologische industrielle und kommerzielle
 vorherrschaft zu verbessern. (-2, +3) (1) (1)
 E: they are increasing their investments to improve their technological industrial and
 commercial predominance.
- D: die usa und japan ergreifen neue initiativen und erhöhen ihre investitionen. (-88,
 +89) (40) (45)
 E: the usa and japan are taking new initiatives and increasing their investments.
- D: es gibt hinweise dafür dass die mittel in ihrer grössenordnung zu gering sind und
 dass sie nicht auf wettbewerbsfähige innovation ausgerichtet sind. (-90, +91,
 -92) (41) (46)
 E: there is evidence that the resources are too small and are not focussed towards
 competitive innovation.
- D: die tätigkeiten in diesem bereich konzentrieren sich auf methoden und instrumente
 und ihre integration in vollständige systeme. (-93, +94, -95, -96) (42) (47)
 E: the activities in this area concentrate on methods and tools and on their intergration
 into complete systems.
- D: die beiden bereiche wurden wegen ihres wachstumspotentials und ihres einflusses auf
 grosse industriezweige ausgewählt. (+97) (43) (48, +49)
 E: the two areas were selected because of their growth potential and impact on large
 industry sectors.
- D: ausserdem war die breite des technologischen spektrums auf das sich ihre entwicklung
 abstützt entscheidend. (+98) (44) (50)
 E: in addition the width of the technological spectrum was based on their development.
- D: die festlegung von zielen für die büroautomation und ihre umsetzung in mög-
 lichkeiten hängen von lokalen kulturellen überlegungen ab. (-99, +100) (45)
 (51)
 E: the definition of objectives for office automation and their translation into opportu-
 nities are dependant upon local cultural considerations.
- D: es gibt keine besonderen beschränkungen für die form einer solchen projektbetei-
 ligung. (+101, -102, -103, -104) (46) (52)
 E: there are no specific restrictions to the form of such coparticipation.
- D: sie kann aus einer kooperation zwischen zwei projekten bestehen. (+105, -106)
 (47) (53)
 E: it can result from cooperation between two projects.

B Die Datei mtmenu.pl

```
% directory for help files

help_directory(help).

% default directories and files for linguistic data

language_directory(g,      '../..//daten/germ').
language_directory(_ --> _, '../..//daten/transfer').
language_directory(e,      '../..//daten/eng').

% gram_file(+Language, +Short, -Comment, -Directory, -File)
%
% - 'Language' indicates the directory according to the predicate
%   'language_directory', in which the 'Directory' is located.
% - 'Short' indicates the data in the 'File' (see list below)
% - 'Directory' is a directory in the directory indicated by 'Language', which
%   contains the 'File'. If 'Directory' is equal to '.', the 'File' has to be
%   in the directory indicated by 'Language'.
% - 'File' indicates the file name. If the name is 'dummy', no data is loaded.
%   If 'File' is a list of atoms, the corresponding list of files is loaded.
%
% Example: gram_file(g, id, 'German ID rules', gpsg, idgerm).
%          --> The file 'idgerm.pl' has to be in the
%              directory '../..//daten/germ/gpsg'.
%
%          gram_file(g, id, 'German ID rules', '.', idgerm).
%          --> The file 'idgerm.pl' has to be in the
%              directory '../..//daten/germ'.
%
% Types of files
%
%   morphological analysis and synthesis
%   (world: 'morpar' and 'morgen', respectively)
%
%   + 'glex':    GPSG-FAS lexicon
%   + 'flex':    FAS-TBox lexicon
%               (also: semantic analysis and generation and transfer)
%   + 'mlex':    morphological lexicon
%   + 'mdata':   morphological data for inflection
%
%   parser (world: 'parser')
%
%   + 'alias':   aliases for GPSGs
%   + 'fcr':     Feature Co-occurrence restrictions for GPSGs
%   + 'id':      Immediate Dominance (ID) rules for GPSGs
%   + 'idc':     comments to the ID rules
%   + 'lp':      Linear Precedence (LP) statements
%   + 'dom':     dominance relations computed from the ID rules
%
%   semantic analysis and generation
%   (world: 'gpsgfas' and 'generator', respectively)
%
%   + 'gdef':    GPSG category specifications
%   + 'hdef':    hybrid (syntactic and semantic) category specifications
%   + 'fdef':    FAS category specifications
%               (also: conceptual analysis and transfer)
```

```

% + 'fas':      FAS grammar (also: transfer)
% + 'idgen':    syntactic grammar (in the format of FAS) for generation
% + 'flex':     FAS-TBox lexicon
%               (also: morphological analysis and synthesis and transfer)
% + 'conf':     case-role tables
% + 'map':      term-rewrite rules for semantic analysis
% + 'pa':       term-rewrite rules for generation
%
% conceptual analysis (world: 'sfasatl')
%
% + 'fdef':     FAS category specifications
%               (also: semantic analysis, generation and transfer)
% + 'adef':     ATL category specifications
% + 'atlgram':  ATL grammar
% + 'atllex':   ATL lexicon
% + 'fasatl':   term-rewrite rules
%
% anaphora evaluation (world: 'eval')
%
% + 'pront':    pronoun table
% + 'edefs':    parameters for the evaluation
%
% transfer (world: 'transfer')
%
% + 'fdef':     FAS category specifications
%               (also: semantic and conceptual analysis, generation)
% + 'fas':      FAS grammar (also: semantic analysis and generation)
% + 'flex':     FAS-TBox lexicon (also: semantic analysis and generation,
%               morphological analysis and synthesis)
% + 'trule':    term-rewrite rules
%
% TBox
%
% + 'tbox':     list of files with TBox definitions
%
gram_file(g, mdata, 'German morph. data',      morph,      data).
gram_file(g, mlex,  'German morph. lexicon',    morph,      morph_lex_g_7).
gram_file(g, glex,  'German GPSG lexicon',      lex,        gpsg_lex_g_7).
gram_file(g, flex,  'German FAS lexicon',        lex,        fas_lex_g_7).
gram_file(g, alias, 'German aliases',          gpsg,       al7).
gram_file(g, gdef,  'German GPSG cat. specs.',   gpsg,       gpsgdefg).
gram_file(g, fcr,   'German FCRs',              gpsg,       fr_g_7).
gram_file(g, id,    'German ID rules',           gpsg,       id_g_7).
gram_file(g, idc,   'German ID comments',        gpsg,       idc_g_7).
gram_file(g, idgen, 'German cf grammar',         gpsg,       dummy).
gram_file(g, lp,    'German LP statements',      gpsg,       lp_g_7).
gram_file(g, dom,   'German DOM relation',       gpsg,       dom_g_7).
gram_file(g, hdef,  'German hybrid cat. specs.', gpsg_fas,  hybdefg5).
gram_file(g, conf,  'German deep cases',         gpsg_fas,  casrole_g_7).
gram_file(g, map,   'German maps GPSG-FAS',     gpsg_fas,  gpfasg7).
gram_file(g, pa,    'German pat.-act. rules',    fas_gpsg,  mapg).
gram_file(g, fdef,  'German FAS cat. specs.',    fas,       fas_cat_g_7).
gram_file(g, fas,   'German FAS grammar',        fas,       fas_clause_g_7).
gram_file(g, fasatl, 'German map rules FAS-ATL', fas_atl,  gatlg7).
gram_file(g, adef,  'German ATL cat. specs',     atl,       atldefg5).
gram_file(g, atlgram, 'German ATL grammar',     atl,       atlgram5).
gram_file(g, atllex, 'German ATL lexicon',       atl,       atllex5).
gram_file(g, pront, 'German pronoun table',     eval,      'pron.fas').

```

```

gram_file(g, edefs, 'German eval. params.', eval, qevaldefs).
gram_file(g, tbox, 'German TBox definitions', tbox, [events, objects,
properties, places])).

gram_file(e, mlex, 'English morph. lexicon', morph, mlexeng).
gram_file(e, mdata, 'English morph. data', morph, engdata).
gram_file(e, glex, 'English syntactic lexicon', lex, gplexe7).
gram_file(e, flex, 'English FAS lexicon', lex, fas_lex_e_7).
gram_file(e, alias, 'English aliases', gpsg, al4).
gram_file(e, gdef, 'English syn. cat. specs.', gpsg, gpdefe7).
gram_file(e, fcr, 'English FCRs', gpsg, fr4).
gram_file(e, id, 'English ID rules', gpsg, id4).
gram_file(e, idc, 'English ID comments', gpsg, idc4).
gram_file(e, idgen, 'English cf grammar', gpsg, gpgram7).
gram_file(e, lp, 'English LP statements', gpsg, lp4).
gram_file(e, dom, 'English DOM relation', gpsg, dom4).
gram_file(e, hdef, 'English hyb. cat. specs.', fas_gpsg, hybdefe7).
gram_file(e, conf, 'English deep cases', fas_gpsg, cre7).
gram_file(e, map, 'English maps FAS-syn.', fas_gpsg, fasgpe7).
gram_file(e, pa, 'English maps FAS-syn.', fas_gpsg, fasgpe7).
gram_file(e, fdef, 'English FAS cat. specs.', fas, fas_cat_e_7).
gram_file(e, fas, 'English FAS grammar', fas, fas_clause_e_7).
gram_file(e, fasatl, 'English maps FAS-ATL', fas_atl, dummy).
gram_file(e, adef, 'English ATL cat. specs.', atl, dummy).
gram_file(e, atlgram, 'English ATL grammar', atl, dummy).
gram_file(e, atllex, 'English ATL lexicon', atl, dummy).
gram_file(e, pront, 'English pronoun table', eval, dummy).
gram_file(e, edefs, 'English eval. params.', eval, dummy).
gram_file(e, tbox, 'English TBox definitions', tbox, []).

gram_file(e --> g, trule, 'Transfer English to German', e_g, transeg1).
gram_file(g --> e, trule, 'Transfer German to English', g_e, tge77).

```

C Die Datei synonyms.pl

```
% List of synonyms for predefined Prolog or MT system commands
%
% synonym(?Synonym, +Predicate, +Arities)
%
% 'Synonym' defines a synonym (atom) for a command, which is
% predefined in the MT or Prolog system.
%
% 'Predicate' has to be the name (atom) of a predicate, which is
% predefined in the MT or Prolog system or another synonym.
%
% 'Arities' is the list of possible arities of the predefined predicate.
%
% WARNING: Please take care of cycles
%
% Do not define direct or indirect cycles, for example synonym(x,x,Arities)
% synonym(x,y,Arities) and synonym(y,x,Arities)

synonym(a, translate, [0]).
synonym(translation, translate, [0]).
synonym(mt, translate, [0]).
synonym(ma, morphological_analysis, [1]).
synonym(p, parse, [0]).
synonym(dcp, parse, [0]).
synonym(parser, parse, [0]).
synonym(parsing, parse, [0]).
synonym(s, semantic_analysis, [0,1]).
synonym(sa, semantic_analysis, [0,1]).
synonym(c, conceptual_analysis, [0,1]).
synonym(ca, conceptual_analysis, [0,1]).
synonym(t, transfer, [0,1]).
synonym(g, generate, [0,1]).
synonym(gen, generate, [0,1]).
synonym(generator, generate, [0,1]).
synonym(generating, generate, [0,1]).
synonym(ms, morphological_synthesis, [3]).

synonym(p, pretty_print, [1,2]).
synonym(d, delete, [1,2]).

synonym(lp, load_parser, [1]).
synonym(lsa, load_semantic_analysis, [1]).
synonym(lca, load_conceptual_analysis, [1]).
synonym(lae, load_anaphora_evaluation, [1]).
synonym(ltr, load_text_representation, [1]).
synonym(lt, load_transfer, [1]).
synonym(lg, load_generator, [1]).

synonym(sw, switch, [1,2]).

synonym(iv, init_vortext, [1]).
synonym(cv, change_vortext, [2]).
synonym(dv, display_vortext, [1]).
synonym(sv1, set_vortext_length, [1]).
synonym(itb, init_text_block, [0]).
```

synonym(ld,	load_data, [1]).
synonym(hist,	history, [0]).
synonym(hc,	history_call, [1]).
synonym(dh,	delete_history, [0]).
synonym(end,	quit, [0]).
synonym(bye,	quit, [0]).
synonym(help,	mt_help, [0]).
synonym(h,	mt_help, [0]).
synonym('?',	mt_help, [0]).
synonym(help,	help_file, [1]).
synonym(h,	help_file, [1]).
synonym('?',	help_file, [1]).
synonym(gc,	generate_category, [4]).
synonym(gencat,	generate_category, [4]).
synonym(r,	reconsult, [1]).
synonym(sp,	save_program, [1]).

D Die Datei noback.pl

```
% List of Prolog or MT system predicates without backtracking
%
% no_backtracking(-System, +Predicate, +Arities)
%
% 'System' is one of 'prolog' or 'mt'.
%
% 'Arities' is the list of possible arities of the predicate.

% MT system predicates

no_backtracking(mt, translate, [0]).
no_backtracking(mt, morphological_analysis, [1]).
no_backtracking(mt, parse, [0]).
no_backtracking(mt, semantic_analysis, [0,1]).
no_backtracking(mt, conceptual_analysis, [0,1]).
no_backtracking(mt, transfer, [0,1]).
no_backtracking(mt, generate, [0,1]).
no_backtracking(mt, morphological_synthesis, [3]).

no_backtracking(mt, pretty_print, [1,2]).
no_backtracking(mt, delete, [1,2]).
no_backtracking(mt, file, [1,2]).

no_backtracking(mt, data, [0]).
no_backtracking(mt, load_parser, [1]).
no_backtracking(mt, load_semantic_analysis, [1]).
no_backtracking(mt, load_conceptual_analysis, [1]).
no_backtracking(mt, load_anaphora_evaluation, [1]).
no_backtracking(mt, load_text_representation, [1]).
no_backtracking(mt, load_transfer, [1]).
no_backtracking(mt, load_generator, [1]).

no_backtracking(mt, switch, [1,2]).
no_backtracking(mt, switches, [0]).

no_backtracking(mt, init_vortex, [1]).
no_backtracking(mt, change_vortex, [2]).
no_backtracking(mt, display_vortex, [1]).
no_backtracking(mt, set_vortex_length, [1]).
no_backtracking(mt, init_text_block, [0]).

no_backtracking(mt, load_data, [1]).
no_backtracking(mt, history, [0]).
no_backtracking(mt, delete_history, [0]).
no_backtracking(mt, shell, [0]).
no_backtracking(mt, mt_help, [0]).
no_backtracking(mt, help_file, [1]).

no_backtracking(mt, reconsult_w, [2]).
no_backtracking(mt, listing_w, [1,2]).
no_backtracking(mt, retractall_w, [1]).

% Prolog predicates
```

```
no_backtracking(prolog, consult, [1]).
no_backtracking(prolog, reconsult, [1]).
no_backtracking(prolog, save_program, [1]).
no_backtracking(prolog, compile, [1]).
no_backtracking(prolog, tell, [1]).
no_backtracking(prolog, see, [1]).
no_backtracking(prolog, told, [0]).
no_backtracking(prolog, seen, [0]).
no_backtracking(prolog, spy, [1]).
no_backtracking(prolog, nospy, [1]).
no_backtracking(prolog, debug, [0]).
no_backtracking(prolog, trace, [0]).
no_backtracking(prolog, nodebug, [0]).
no_backtracking(prolog, notrace, [0]).
no_backtracking(prolog, assert, [1]).
no_backtracking(prolog, asserta, [1]).
no_backtracking(prolog, assertz, [1]).
no_backtracking(prolog, retract, [1]).
```

Literaturverzeichnis

- [Arnold et al. 86] D.J. Arnold, S. Krauwer, M. Rosner, L. de Tombe, G.B. Varile, *The $\langle C, A \rangle, T$ Framework in EUROTRA: A Theoretically Committed Notation for MT*, in: Proceedings 11th COLING-86, Bonn 1986, S. 297-303
- [Arnold/des Tombe 87] D.J. Arnold, L. des Tombe, *Basic theory and methodology in EUROTRA*, in: [Nirenburg 87], S. 114-135
- [Bátori 89] I.S. Bátori, *Systemarchitektur der Sprachverstehenssysteme* in: I.S. Bátori, W. Lenders, W. Putschke (Hrsg.), *Computerlinguistik – Ein internationales Handbuch zur computergestützten Sprachforschung und ihrer Anwendungen*, de Gruyter, Berlin New York 1989, S. 311-316
- [Bläsius/Bürckert 87] K.H. Bläsius, H.-J. Bürckert, *Deduktionssysteme, Automatisierung des logischen Denkens*, Oldenbourg, München Wien 1987
- [Boitet/Gerber 84] C. Boitet, R. Gerber, *Expert Systems and other Techniques in MT Systems*, in: Proceedings 10th COLING-84, Stanford 1984, S. ??
- [Briscoe et al. 87] T. Briscoe, C. Grover, B. Boguraev, J. Carroll, *A Formalism and Environment for the Development of a Large Grammar of English*, in: Proceedings 10th IJCAI-87, Mailand 1987, S. 703-708
- [Busemann 83a] S. Busemann, *Oberflächentransformationen bei der automatischen Generierung geschriebener deutscher Sprache. Entwurf und Implementierung des modularen und anpaßbaren Systems SUTRA*, Diplomarbeit, FB Informatik, Universität Hamburg 1983
- [Busemann 83b] S. Busemann, *Oberflächentransformationen bei der Generierung geschriebener deutscher Sprache*, in: Proceedings 7th German Workshop on Artificial Intelligence (GWAI-83), Dassel/Solling 1983, Springer, S. 90-99
- [Busemann 92] S. Busemann, *Generierung natürlicher Sprache mit Generalisierten Phrasenstrukturgrammatiken*, Informatik Fachberichte 313, Springer, Berlin 1992
- [Busemann/Hauenschild 88] S. Busemann, Ch. Hauenschild, *A Constructive View of GPSG or How to Make it Work*, in: Proceedings 12th COLING-88, Budapest 1988, S. 77-82
- [Busemann/Hauenschild 89] S. Busemann, Ch. Hauenschild, *From FAS Representations to GPSG Structures*, in: S. Busemann, Ch. Hauenschild und C. Umbach (Hrsg.), *Views of the Syntax/Semantics Interface*, Proceedings Workshop GPSG and Semantics, KIT-Report 74, Technische Universität Berlin 1989, S. 17-43
- [Carbonell/Tomita 87] J. G. Carbonell, M. Tomita, *Knowledge-based machine translation, the CMU approach*, in: [Nirenburg 87], S. 68-89
- [Davis/King 77] R. Davis, J. King, *An Overview of Production Systems*, in: E.W. Elcock, D. Michie (Hrsg.), *Machine Intelligence 8*, Ellis Horwood, Chichester 1977, S. 300-332

- [Dershowitz 82] N. Dershowitz, *Orderings for Term-Rewriting Systems*, Theoretical Computer Science 17 (1982), North-Holland, S. 279-301
- [Dershowitz 85] N. Dershowitz, *Termination*, in: G. Goos, J. Hartmanis (Hrsg.), *Rewriting Techniques and Applications*, LNCS 202, Dijon 1985, S. 180-224
- [Dörfler 77] W. Dörfler, *Mathematik für Informatiker, Band 1 Finite Methoden und Algebra*, Hanser, München 1977
- [Dörre/Eisele 89] J. Dörre, A. Eisele, *Determining Consistency of Feature Terms with Distributed Disjunctions*, in: Proceedings 13th German Workshop on Artificial Intelligence (GWAI-89), Eringerfeld 1989, Springer, S. 270-279
- [Dörre/Eisele 90] J. Dörre, A. Eisele, *Feature Logic with Disjunctive Unification*, in: Proceedings 13th COLING-90 (Vol. 2), Helsinki 1990, S. 100-105
- [Dörre/Momma 85] J. Dörre, S. Momma, *Modifikationen des Earley-Algorithmus und ihre Verwendung für ID/LP-Grammatiken*, Manuskript, Institut für maschinelle Sprachverarbeitung, Universität Stuttgart 1985
- [Drewes 89] F. Drewes, *Termersetzungssysteme: Termination und Komplexität*, Diplomarbeit, FB Mathematik und Informatik, Universität Bremen 1989
- [Drewes/Lautemann 90] Fr. Drewes, Cl. Lautemann, *Incremental Termination Proofs and the Length of Derivations*, Bericht Nr. 7/90, FB Mathematik und Informatik, Universität Bremen 1990
- [Dunker 93] G. Dunker, *Beschreibung des GPSG-Übersetzers, des FAS-Grammatik Übersetzers und des Lexikon Übersetzers als Werkzeuge für das experimentelle MÜ-System des Projekts KIT-FAST*, Manuskript, Technische Universität Berlin 1993, Projektgruppe KIT, Sekr. FR 5-12, Franklinstr. 28/29, D-10587 Berlin
- [Dunker/Umbach 93] G. Dunker, C. Umbach, *Verfahren zur Anaphernresolution in KIT-FAST*, KIT-Interner Arbeitsbericht in Vorbereitung, Technische Universität Berlin 1993
- [Ehrig/Mahr 85] H. Ehrig, B. Mahr, *Fundamentals of Algebraic Specification 1, Equations and Initial Semantics*, EATCS 6, Springer, Berlin 1985
- [Ehrig/Mahr 89] H. Ehrig, B. Mahr, *Fundamentals of Algebraic Specification 2, Module Specifications and Constraints*, EATCS 21, Springer, Berlin 1989
- [Eisele/Dörre 86] A. Eisele, J. Dörre, *A Lexical Functional Grammar System in Prolog*, in: Proceedings 11th COLING-86, Bonn 1986, S. 551-553
- [Eisele/Dörre 88] A. Eisele, J. Dörre, *Unification of disjunctive feature descriptions*, in: Proceedings 26th ACL, Buffalo(NY) 1988
- [Emele 89] M. Emele, *A Typed Feature Structure Unification-based Approach to Generation*, Internes Papier NLC 88-32, ATR, Osaka 1989
- [Emele 91] M. Emele, *Unification with Lazy Non-Redundant Copying*, in: Proceedings 29th ACL, Berkeley(CA) 1991, S. 323-330

- [Emele/Zajac 89] M. Emele, R. Zajac, *RETIF: A Rewriting System for Typed Feature Structures*, Technical Report TR-I-0071, ATR, Osaka 1989
- [Emele/Zajac 90] M. Emele, R. Zajac, *Typed Unification Grammars*, in: Proceedings 13th COLING-90 (Vol. 3), Helsinki 1990, S. 293-298
- [Euler 91] L. Euler, *Eigenschaften von Unifikationsformalismen*, ASL-Bericht Nr. ASL-TR-23-91/UHH, Universität Hamburg 1991
- [Evans 87] R. Evans, *Theoretical and Computational Interpretations of Generalized Phrase Structure Grammar*, Cognitive Science Research Paper CSRP 085, University of Sussex 1987
- [Gazdar et al. 85] G. Gazdar, E. Klein, G. Pullum und I. Sag, *Generalized Phrase Structure Grammar*, Blackwell, Oxford 1985
- [Goodman/Nirenburg 91] K. Goodman, S. Nirenburg (Hrsg.), *The KBMT Project: A Case Study in Knowledge-Based Machine Translation*, Morgan Kaufmann, San Mateo/CA 1991
- [Gräf 90] A. Gräf, *Efficient Pattern Matching for Term Rewriting*, Informatik-Bericht Nr. 3/90, Universität Mainz 1990
- [Grosz/Sidner 86] B.J. Grosz, C.L. Sidner, *Attention, Intentions, and the Structure of Discourse*, in: Computational Linguistics Vol. 12 No. 3 (1986), S. 175-204
- [Hauenschild 86] Ch. Hauenschild, *KIT/NASEV oder die Problematik des Transfers bei der maschinellen Sprachübersetzung*, in: I. Bátori, H.J. Weber (Hrsg.): *Neue Ansätze in Maschinellem Sprachübersetzung: Wissensrepräsentation und Textbezug*, Niemeyer, Tübingen 1986, S. 167-195
- [Hauenschild 87] Ch. Hauenschild, *Textlinguistische Probleme der maschinellen Übersetzung*, in: R. Arntz (Hrsg.), *Textlinguistik und Fachsprache, Akten des internationalen übersetzungswissenschaftlichen AILA-Symposiums 13.-16. April 1987 in Hildesheim*, Olms, Hildesheim 1988
- [Hauenschild 88] Ch. Hauenschild, *Discourse Structure – Some Implications for Machine Translation*, in: D. Maxwell, K. Schubert, A.P.M. Witkam (Hrsg.), *New Directions in Machine Translation*, Proceedings of the Conference, Foris, Dordrecht 1988, S. 145-156
- [Hauenschild 91] Ch. Hauenschild, *Anaphern-Interpretation in der Maschinellen Übersetzung*, Zeitschrift für Literaturwissenschaft und Linguistik 84 (1991), Vandenhoeck & Ruprecht, S. 50-66
- [Hauenschild et al. 78] Ch. Hauenschild, E. Huckert, R. Maier, *SALAT: Entwurf eines automatischen Übersetzungssystems*, in: Sprache und Datenverarbeitung 2 (1978), S. 126-152
- [Hauenschild et al. 79] Ch. Hauenschild, E. Huckert, R. Maier, *SALAT: Machine Translation Via Semantic Representation*, in: R. Bäuerle, U. Egli, A. von Stechow (Hrsg.): *Semantics from Different Points of View*, Springer, Berlin 1979, S. 324-352

- [Hauenschild/Busemann 88] Ch. Hauenschild, S. Busemann, *A constructive version of GPSG for machine translation*, in: [Steiner et al. 88a], S. 216-238
- [Hauenschild/Umbach 88] Ch. Hauenschild, C. Umbach, *Funktor-Argument-Struktur, Die satzsemantische Repräsentations- und Transferebene im Projekt KIT-FAST*, in: [Schütz 88], S. 16-35
- [Heid et al. 88] U. Heid, K. Netter, J. Wedekind, *Zur Funktionsweise des Transfers auf f-Strukturen*, in: [Schütz 88], S. 8-15
- [Henschel 86a] R. Henschel, *Ein Filter für die automatische syntaktische Analyse als Graphen-Transformationssystem*, Dissertation A, Zentralinstitut für Sprachwissenschaft, Berlin 1986
- [Henschel 86b] R. Henschel, *Das Graphen-Transformationssystem GRACOLI*, in: Mitteilungen zur automatischen Sprachverarbeitung September 1986, Zentralinstitut für Sprachwissenschaft, Berlin 1986, S. 1-32
- [Hoffmann/Plump 88] B. Hoffmann, D. Plump, *Jungle Evaluation for Efficient Term Rewriting*, Informatik Bericht Nr. 4/88, Universität Bremen 1988
- [Hoppe et al. 93] T. Hoppe, C. Kindermann, J. Quantz, A. Schmiedel, M. Fischer, *BACK V5 – Tutorial & Manual*, KIT-Report 100, Technische Universität Berlin 1993
- [Huckert 79] E. Huckert, *Automatische Synthese des Französischen aus einer logischen Basis*, AQ-Verlag, Dudweiler bei Saarbrücken 1979
- [Huet 77] G. Huet, *Confluent Reductions: Abstract Properties and Applications to Term Rewriting*, in: Proceedings 18th IEEE Symposium on Foundations of Computer Science, 1977
- [Huet/Oppen 80] G. Huet, D. Oppen, *Equations and Rewrite Rules*, in: R.V. Book (Hrsg.), *Formal Language Theory, Perspectives and Open Problems*, Academic Press 1980, S. 349-405
- [Hußmann 89] H. Hußmann, *Nichtdeterministische algebraische Spezifikation*, Dissertation, Fakultät für Mathematik und Informatik, Universität Passau 1989
- [Kaplan/Bresnan 82] R.M. Kaplan, J. Bresnan, *Lexical Functional Grammar – A Formal System for Grammatical Representation*, in: J. Bresnan (Hrsg.), *The Mental Representation of Grammatical Relations*, Cambridge, MA., MIT Press 1982, S. 173-281
- [Kaplan et al. 89] R.M. Kaplan, K. Netter, J. Wedekind, A. Zaenen, *Translation by Structural Correspondences*, in: Proceedings 4th Conference of the European Chapter of the ACL, Manchester 1989, S. 272-281
- [Kaplan/Maxwell 88] R.M. Kaplan, J.T. Maxwell III, *An Algorithm for Functional Uncertainty*, in: Proceedings 12th COLING-88, Budapest 1988, S. 297-302
- [Kaplan/Zaenen 89] R.M. Kaplan, A. Zaenen, *Long-distance dependencies, constituent structure, and functional uncertainty*, in: M. Baltin, A. Kroch (Hrsg.), *Alternative Conceptions of Phrase Structure*, University Press, Chicago 1989, S. 17-42

- [Kay 83] M. Kay, *When Metarules are not Metarules*, in: K. Sparck Jones, Y. Wilks (Hrsg.), *Automatic Natural Language Parsing*, Ellis Horwood, West Sussex 1983, S. 94-117
- [Kilbury 84] J. Kilbury, *Earley-basierte Algorithmen für direktes Parsen mit ID/ LP-Grammatiken*, KIT-Report 16, Technische Universität Berlin 1984
- [KIT-FAST 91] Projekt KIT-FAST: S. Busemann, T. Eckard, M. Garry, Ch. Hauenschild, B. Mahr, A. MacLarnon, S. Preuß, B. Schmitz, C. Umbach, W. Weisweber, Ch. Werner-Meier, L. Wilson, E. Ziegler, *Schlußbericht des Berliner Projekts der EUROTRA-D-Begleitforschung "Transfer und Generierung auf satzsemantischer Basis"*, KIT-Report 88, Technische Universität Berlin 1991
- [KIT-FAST 93] Projekt KIT-FAST: Ch. Hauenschild, B. Mahr, S. Preuß, B. Schmitz, C. Umbach, W. Weisweber, L. Beheshty, G. Dunker, M. Rickard, Ch. Werner-Meier, E. Ziegler, *Schlußbericht des Berliner Projekts der EUROTRA-D-Begleitforschung "Anapherninterpretation in der maschinellen Übersetzung"*, KIT-Report 108, Technische Universität Berlin 1993
- [Knuth/Bendix 70] D. Knuth, P. Bendix, *Simple Word Problems in Universal Algebras*, in: J. Leech (Hrsg.), *Computational Problems in Abstract Algebra*, Pergamon Press 1970, S. 263-297
- [Kogure 92] K. Kogure, *A Treatment of Negative Descriptions of Typed Feature Structures*, in: Proceedings 14th COLING-92, Nantes 1992, S. 380-386
- [Koutsoudas 76] A. Koutsoudas (Hrsg.), *The Application and Ordering of Grammatical Rules*, The Hague 1976
- [Mahr/Umbach 90] B. Mahr, C. Umbach, *Functor-Argument Structures for the Meaning of Natural Language Sentences and Their Formal Interpretation*, in: K.H. Bläsius, U. Hedtstück, C. Rollinger (Hrsg.), *Sorts and Types in Artificial Intelligence*, Lecture Notes in Artificial Intelligence, Springer, Berlin 1990, S. 286-304
- [Nagao 87] M. Nagao, *Role of structural transformation in a machine translation system*, in: [Nirenburg 87], S. 262-277
- [Nakano 91] M. Nakano, *Constraint Projection: An Efficient Treatment of Disjunctive Feature Descriptions*, in: Proceedings 29th ACL, Berkeley(CA) 1991, S. 307-314
- [Naumann 88] S. Naumann, *Generalisierte Phrasenstrukturgrammatik: Parsingstrategien, Regelorganisation und Unifikation*, Niemeyer, Tübingen 1988
- [Nirenburg 87] S. Nirenburg (Hrsg.), *Machine Translation: Theoretical and Methodological Issues*, University Press, Cambridge/England 1987
- [Nirenburg et al. 87] S. Nirenburg, V. Raskin, A.B. Tucker, *The structure of interlingua in TRANSLATOR*, in: [Nirenburg 87], S. 90-113

- [Nirenburg et al. 92] S. Nirenburg, J. Carbonell, M. Tomita, K. Goodman (Hrsg.), *Machine Translation: A Knowledge-Based Approach*, Morgan Kaufmann, San Mateo/CA 1992
- [Peltason et al. 89] C. Peltason, A. Schmiedel, C. Kindermann, J. Quantz, *The BACK System Revisited*, KIT-Report 75, Technische Universität Berlin 1989
- [Phillips/Thompson 87] J.D. Phillips, H.S. Thompson, *A Parser for Generalized Phrase Structure Grammars*, in: N. Haddock, E. Klein, G. Morril (Hrsg.), *Working Papers in Cognitive Science, Volume I. Categorical Grammar, Unification Grammar and Parsing*, Centre for Cognitive Science, University of Edinburgh 1987, S. 115-136
- [Preuß 87] S. Preuß, *GPSG-Syntax für ein Fragment des Deutschen*, KIT-Interner Arbeitsbericht 20, Technische Universität Berlin 1987
- [Preuß 89] S. Preuß, *Koordination und Kongruenz in einer Verallgemeinerten Phrasenstrukturgrammatik*, Magisterarbeit, FB Linguistik der Technischen Universität Berlin 1989
- [Preuß et al. 92] S. Preuß, B. Schmitz, C. Hauenschild, *Anaphora Resolution Based on Semantic and Conceptual Knowledge*, in: S. Preuß, B. Schmitz (Hrsg.), *Text Representation and Domain Modelling – Ideas From Linguistics and AI*, Proceedings des Workshops, KIT-Report 97, Technische Universität Berlin 1992, S. 1-13
- [Preuß et al. 93] S. Preuß, B. Schmitz, C. Hauenschild, C. Umbach, *Anaphora Resolution in Machine Translation*, erscheint in: W. Ramm, P. Schmidt, J. Schütz (Hrsg.), *Studies in Machine Translation and Natural Language Processing*, Volume on “Discourse in Machine Translation”
- [Quantz/Kindermann 90] J. Quantz, C. Kindermann, *Implementation of the BACK System Version 4*, KIT-Report 78, Technische Universität Berlin 1990
- [Russell et al. 91] G. Russell, A. Ballim, D. Estival, S. Warwick-Amstrong, *A Language for the Statement of Binary Relations over Feature Structures*, in: Proceedings 5th Conference of the European Chapter of the ACL, Berlin 1991, S. 287-292
- [Schmitz et al. 92] B. Schmitz, S. Preuß, C. Hauenschild, *Textrepräsentation und Hintergrundwissen für die Anaphernresolution im Maschinellen Übersetzungssystem KIT-FAST*, KIT-Report 93, Technische Universität Berlin 1992
- [Schütz 88] J. Schütz (Hrsg.), *Workshop Semantik und Transfer*, EUROTRA-D Working Papers No. 6, IAI, Universität des Saarlandes, Saarbrücken 1988
- [Schütz/Sharp 88] J. Schütz, R. Sharp, *CAT2-R – Komplexität eines Formalismus für multilinguale MÜ*, in: [Schütz 88], S. 36-45
- [Sgall et al. 73] P. Sgall, E. Hajičová, E. Benešová, *Topic, Focus and Generative Semantics*, Scriptor, Kronberg 1973

- [Sharp 88] R. Sharp, *CAT2 – Implementing a Formalism for Multi-Lingual MT*, in: Proceedings 2nd International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages, CMU Pittsburg 1988
- [Shieber 84] S.M. Shieber, *Direct Parsing of ID/LP-Grammars*, in: Linguistics and Philosophy 7, 1984, S. 135-154
- [Sikkel 92] K. Sikkel, *Feature Structure Unification Algorithms*, Manuskript, Computer Science Department, Universität Twente, Enschede 1992
- [Steiner et al. 88a] E. Steiner, P. Schmidt, C. Zellinsky-Wibbelt, *From Syntax to Semantics. Insights from Machine Translation*, Frances Pinter, London 1988
- [Steiner et al. 88b] E. Steiner, U. Eckert, B. Roth, J. Winter-Thielen, *The Development of the EUROTRA-D System of Semantic Relations*, in: [Steiner et al. 88a] S. 40-104
- [Sterling/Shapiro 86] L. Sterling, E. Shapiro, *The Art of Prolog*, MIT Press, Cambridge/MA 1986
- [Tomabechi 92] H. Tomabechi, *Quasi-Destructive Graph Unification with Structure-Sharing*, in: Proceedings 14th COLING-92, Nantes 1992, S. 440-446
- [Tremblay/Sorenson 85] J.-P. Tremblay, P.G. Sorenson, *The Theory and Practice of Compiler Writing*, McGraw-Hill, New York 1985
- [Tsuji 86] J.-I. Tsuji, *Future Directions of Machine Translation*, in: Proceedings 11th COLING-86, Bonn 1986, S. 655-668
- [Uszkoreit 84] H. Uszkoreit, *Word Order and Constituent Structure in German*, Ph.D. Dissertation, University of Texas, Austin 1984
- [van Noord 90] G. van Noord, *Reversible Unification Based Machine Translation*, in: Proceedings 13th COLING-90 (Vol. 2), Helsinki 1990, S. 299-304
- [Véronis 92] J. Véronis, *Disjunctive Feature Structures as Hypergraphs*, in: Proceedings 14th COLING-92, Nantes 1992, S. 498-504
- [Wedekind 86] J. Wedekind, *A Concept of Derivation for LFG*, in: Proceedings 11th COLING-86, Bonn 1986, S. 487-493
- [Wedekind 88] J. Wedekind, *Generation as Structure Driven Derivation*, in: Proceedings 12th COLING-88, Budapest 1988, S. 732-737
- [Weisweber 87a] W. Weisweber, *Ein Dominanz-Chart-Parser für Generalisierte Phrasenstrukturgrammatiken*, KIT-Report 45, Technische Universität Berlin 1987
- [Weisweber 87b] W. Weisweber, *Ein Parser für direktes Parsen mit ID/LP-Grammatiken und komplexen Kategorien*, KIT-Internes Manuskript, Technische Universität Berlin 1987
- [Weisweber 88] W. Weisweber, *Using Constraints in a Constructive Version of GPSG*, in: Proceedings 12th COLING-88, Budapest 1988, S. 738-743

- [Weisweber 89] W. Weisweber, *Transfer in Machine Translation by Non-Confluent Term-Rewrite Systems*, in: Proceedings 13th German Workshop on Artificial Intelligence (GWAI-89), Eringerfeld 1989, Springer, S. 264-269
- [Weisweber 92] W. Weisweber, *Term-Rewriting as a Basis for a Uniform Architecture in Machine Translation*, in: Proceedings 14th COLING-92, Nantes 1992, S. 777-783
- [Weisweber 94a] W. Weisweber, *Termersetzung als Basis für eine einheitliche Architektur in der maschinellen Sprachübersetzung*, Dissertation, Fachbereich Informatik, Technische Universität Berlin 1993, Sprache und Information 28, Niemeyer Verlag, Tübingen 1994
- [Weisweber 94b] W. Weisweber *Benutzerhandbuch des Editors für Termersetzungssysteme (TED) im experimentellen MÜ-System des Projekts KIT-FAST*, Manuskript, Technische Universität Berlin 1994, Projektgruppe KIT, Sekr. FR 5-12, Franklinstr. 28/29, D-10587 Berlin
- [Weisweber/Hauenschild 90] W. Weisweber, Ch. Hauenschild, *A Model of Multi-Level Transfer for Machine Translation and Its Partial Realization*, KIT-Report 77, Technische Universität Berlin 1990 und erscheint in: Proceedings Seminar *Computers & Translation '89*, Tbilisi 1989
- [Weisweber/Preuß 92] W. Weisweber, S. Preuß, *Direct Parsing with Metarules* in: Proceedings 14th COLING-92, Nantes 1992, S. 1111-1115
- [Wernecke 92] W. Wernecke, *THOR – A System for Computing Equational Theories*, IWBS Report 226, IBM Wissenschaftliches Zentrum, Heidelberg 1992
- [Zajac 89] R. Zajac, *A Transfer Model Using a Typed Feature Structure Rewriting System with Inheritance*, in: Proceedings 27th Annual Meeting of the ACL, Vancouver 1989, S. 1-6
- [Zajac 90] R. Zajac, *A relational approach to translation*, in: Proceedings 3rd International Conference on Theoretical and Methodological Issues in Machine Translation of Natural Languages, Austin 1990
- [Zajac 91] R. Zajac, *A Uniform Architecture for Parsing, Generation and Transfer*, in: T. Strzalkowski (Hrsg.), Proceedings Workshop on Reversible Grammar in Natural Language Processing, Berkeley 1991, S. 71-80
- [Zellinsky-Wibbelt 88] C. Zellinsky-Wibbelt, *From Cognitive Grammar to the Generation of Semantic Interpretation in Machine Translation*, in: [Steiner et al. 88a], S. 105-132

Verzeichnis der Abkürzungen

ABox	assertionales Wissen
AP	Agreement Prinzip
AS	Ausgangssprache
ATL	ABox-Tell-Language
BACK	Berlin Advanced Knowledge Representation System
BTL	BACK-Term-Language
CAP	Control Agreement Principle
CAT	Constructors, Atoms and T-Rules
DAG	gerichteter azyklischer Graph
FAS	Funktor-Argument-Struktur
FAST	Funktor-Argument-Struktur für den Transfer
FCR	Feature Co-occurrence Restriction
FFP	Foot Feature Principle
FSD	Feature Specification Default
GPST	Generalisierte Phrasenstrukturgrammatik
HFC	Head Feature Convention
HPSG	Head-Driven Phrase Structure Grammar
ID	Immediate Dominance
KI	Künstliche Intelligenz
KIT	Künstliche Intelligenz und Textverstehen
LFG	Lexical Functional Grammar
LP	Linear Precedence
MIP	Merkmalinstantiierungsprinzip
MÜ	maschinelle Sprachübersetzung
NASEV	neue Analyse- und Syntheseverfahren für die MÜ
NLP	Verarbeitung natürlicher Sprache
PS	Produktionssystem
RETIF	Rewriting System for Typed Feature Structures
SALAT	System for Automatic Analysis and Translation
TBox	terminologisches Wissen
TE	Termersetzung
TE-Regeln	Termersetzungsregeln
TES	Termersetzungssystem
TFS	Typed Feature Structure
ZS	Zielsprache

Stichwortverzeichnis

-Argument, 36
+Argument, 35
?Argument, 36
==>/2, 51
adef, 62
adj/5, 54
alias (Parameter), 108
alias/2, 56
alt/4, 48
ap (Parameter), 113
assert_w/2, 78
asserta_w/2, 78
assertz_w/2, 78
atlgam, 63
atllex, 63
call_w/2, 77
card_features/1, 58
category_def/3, 46
category_position/2, 58
change_vortext/2, 115
conceptual_analysis/ $\{0,1\}$, 85
conf, 61
constr (Parameter), 109
consult_w/2, 77
count (Parameter), 113
cr_noun/5, 62
cr_verb/5, 62
ctr_dec/2, 79
ctr_inc/2, 79
ctr_is/2, 79
ctr_set/2, 79
data/0, 99
default (Parameter), 107
default/2, 48
delete/ $\{0,1\}$, 95
detrel/5, 53
direction/1, 47
display_vortext/1, 116
display (Parameter), 106
dom/2, 60
edefs, 64
edge (Parameter), 111
edge/6, 14
eq/3, 15
exit/0, 70
extend (Parameter), 106
fasatl, 63
fas, 61, 65
fcr/3, 57
fdef, 61, 62, 65
feature_def/3, 47
file/1, 96
fip (Parameter), 107
flex, 52, 61, 65
flex/3, 51
foot_range/1, 58
gdef, 60
generate_category/4, 74
generate/ $\{0,1\}$, 90
getAbstand/1, 64
getGrenzwert/1, 64
gewicht/2, 64
glex/4, 51
gram_file/5, 98
halt/0, 72
hdef, 61
head_spec/3, 59
help_directory/1, 97
help_file/1, 70
hfc (Parameter), 114
history_call/0, 71
history_delete/0, 71
history/0, 71
id_comment/2, 59
idc, 59
idgen, 61
idtrace (Parameter), 109
id/3, 58
init_text_block/0, 116
init_vortext/1, 115
interpret/0, 70
krit_modes/1, 64
kriterien/1, 64
language_directory/2, 98
last_rule/2, 58
listing_w/1, 78
listing_w/2, 78
load_anaphora_evaluation/1, 102
load_data/1, 71
load_generator/1, 104

- load_parser/1, 100
- load_semantic_analysis/1, 101
- load_text_representation/1, 103
- load_transfer/1, 103
- lp, 59
- map_alias/2, 60
- map, 62
- mdata, 53
- meta/4, 50
- meta/5, 57
- mlex/3, 52
- morphological_analysis/1, 81
- morphological_synthesis/3, 92
- mt_help/0, 70
- mtmenu.pl, 97
- neq/3, 15
- no_backtracking/3, 73
- noun/5, 54
- ordered_prec_set/1, 48
- parser (Parameter), 110
- parse/0, 81
- pa, 62
- possflex/4, 53
- possible_values/2, 58
- prec/2, 47
- pretty_print/{1,2}, 95
- prn/5, 55
- pronspectab_g/5/2, 63
- pront, 63
- propose (Parameter), 112
- quant/5, 54
- quit/0, 72
- reconsult_w/2, 77
- refl/5, 55
- retract_w/2, 78
- retractall_w/2, 78
- rewrite_rule/3, 50
- rw_cats/3, 49
- semantic_analysis/{0,1}, 83
- set_vortextlength/1, 116
- shell/0, 72
- specification_feature/3, 59
- switches/0, 105
 - Beispielanfrage, 106
- switch/{1,2}, 105
- synonym/3, 72
- syntax (Parameter), 114
- tbox, 65

- time (Parameter), 107
- trans_lp/2, 59
- transfer/{0,1}, 88
- translate/0, 92
- trule, 65
- verb/5, 56
- vseg/5, 55
- Ableitungsbaum
 - Ausgabe, 74
- ABox, 10
- ABox-Tell-Language, 10
- Abstandswert, 19
- Adjektiv, 54
- Adjektivflexion, 14
- Agreement-Prinzip, 59, 113
- Alias, 56
- Alias-Definition, 45
- allgemeine MÜ-System-Parameter, 106
- Anapherninterpretation, 19
 - Daten, 63, 99
 - Kurzbeschreibung, 19
 - Laden, 102
 - MÜ-System-Parameter, 114
- Anwendungsreihenfolge, 18
- AP, 59, 113
- Argumentmodus, 35
- Argumentschema, 43
- Artikel, 53
- assertionales Wissen, 10
- ATL-Kategorie, 46
- ATL-Kategorienterm, 46, 62
- ATL-Lexikon, 63
- ATL-Regel, 63
- Ausgabe
 - Ableitungsbaum, 74
 - von Repräsentationen, 95
- BACK-System, 1, 10
 - Daten, 65, 99
- Backtracking
 - Ausschluß eines Prädikats, 73
- Baum
 - lokal, 5
- Beispielanfrage
 - switches/0, 106
- Beispiellauf
 - edge = on, 111
 - fip = on, 108

- parser = on, 110
 - propose = on, 112
- CAP, 5
- Constraints, 15
 - Auswertung, 16
- Control-Agreement-Prinzip, 5
- Daten
 - Anapherninterpretation, 99
 - BACK-System, 99
 - Generierung, 98
 - konzeptuelle Analyse, 99
 - Laden, 96
 - morphologische Analyse, 98
 - morphologische Synthese, 98
 - Parser, 98
 - semantische Analyse, 98
 - syntaktische Analyse, 98
 - Transfer, 99
- DCP
 - Kurzbeschreibung, 14
- Defaultregel, 18
- Deklination
 - von Adjektiven, 14
 - von Nomen, 14
 - von Substantiven, 14
- Dominanz-Chart-Parser, 14
 - Kurzbeschreibung, 14
- Dominanzmenge, 14
- Dominanzrelation, 14, 60
- Extension, 15
- FAS-Ableitungsbaum, 34
- FAS-Grammatik, 34
- FAS-Kategorie, 45
- FAS-Kategoriendefinition, 34
- FAS-Kategorienterm, 45, 61, 62, 65
- FAS-Lexikon, 51
- FAS-Regel, 61, 65
- FAS-Term, 34
- FCR, 5, 15, 57
- Feature-Co-occurrence-Restriction, 5, 15, 57
- Feature-Specification-Default, 5
- FFP, 5
- Flexion
 - von Adjektiven, 14
 - von Nomen, 14
 - von Personalpronomen, 13
 - von Substantiven, 14
 - von Verben, 13
- Flexionsmorphem, 13
- Flexionsschema, 13
- Foot-Feature-Prinzip, 5
- FOOT-Merkmal, 58
- FSD, 5
- Generierung, 61, 62, 90
 - Beispiellauf, 91
 - Daten, 60, 98
 - Laden, 104
- GPS-Grammatik, 34
- GPSG-Kategorie, 45, 60
- GPSG-Kategorienterm, 45, 60
- Grammatik
 - im ID/LP-Format, 4
 - im Projekt KIT-FAST, 7, 21
- Grenzwert, 19
- Grundterm, 17
- Hauptkategorie, 4
- Head, 59
- Head-Feature-Convention, 5, 59, 114
- HFC, 5, 59, 114
- hybrider Kategorienterm, 61
- ID-Regel, 4, 58
- ID/LP-Grammatik, 4
- Immediate-Dominance-Regel, 4, 58
- Implementierung, 35
- Interpretierer
 - Implementation, 38
- irreduzibel, 50
- Kürzel, 98
 - adef, 62
 - alias, 56
 - atlgram, 63
 - atllex, 63
 - conf, 61
 - dom, 60
 - edefs, 64
 - fasatl, 63
 - fas, 61, 65
 - fcr, 57
 - fdef, 61, 62, 65

- `flex`, 52, 61, 65
- `gdef`, 60
- `glex`, 51
- `hdef`, 61
- `idc`, 59
- `idgen`, 61
- `id`, 57
- `lp`, 59
- `map`, 62
- `mdata`, 53
- `mlex`, 52
- `pa`, 62
- `pront`, 63
- `tbox`, 65
- `trule`, 65
- Kante der Chart, 14
- Kasus-Rollen-Tabelle, 62
- Kategorie
 - Daten für komplexe ..., 46
 - komplex, 44
- Kategorie
 - ATL, 46
 - FAS, 45
 - GPSG, 45
- Komparation
 - von Adjektiven, 14
- komplexe Kategorie, 44, 46
 - Daten, 46
- Konjugation
 - von Verben, 13
- kontextfreie Grammatik
 - Daten, 50
- kontextfreie FAS-Regel, 34
- kontextfreie Regel, 50, 51
- konzeptuelle Analyse, 23, 63, 85
 - Beispiellauf, 87
 - Daten, 62, 99
 - Laden, 102
- Kriterien, 64
- Löschen
 - von Repräsentationen, 95
- Laden
 - Anapherninterpretation, 102
 - Generierung, 104
 - konzeptuelle Analyse, 102
 - semantische Analyse, 101
 - syntaktische Analyse, 100
- Textrepräsentation, 103
- Transfer, 103
 - von linguistischen Daten, 96, 99
- Lexikon, 50, 51
- Linear-Precedence-Aussage, 4, 15, 59
- linguistische Daten
 - Laden, 96, 99
- lokaler Baum, 5
- LP-Aussage, 4, 15, 59
- MÜ-System-Befehle, 70
- MÜ-System-Interpretierer, 68
- MÜ-System-Parameter
 - `alias`, 108
 - `ap`, 113
 - `constr`, 109
 - `count`, 113
 - `default`, 107
 - `display`, 106
 - `edge`, 111
 - `extend`, 106
 - `fip`, 107
 - `hfc`, 114
 - `idtrace`, 109
 - `parser`, 110
 - `propose`, 112
 - `syntax`, 114
 - `time`, 107
 - allgemein, 106
 - Anapherninterpretation, 114
 - Parser, 107
 - Termersetzungsverfahren, 114
 - verändern, 105
- MÜ-System-Parameter `edge`
 - Beispiellauf, 111
- MÜ-System-Parameter `fip`
 - Beispiellauf, 108
- MÜ-System-Parameter `parser`
 - Beispiellauf, 110
- MÜ-System-Parameter `propose`
 - Beispiellauf, 112
- Merkmalinstantiierungsprinzip, 5
- Metaregel, 4, 50, 57
- minimal, 50
- morphologische Analyse, 13, 81
 - Beispiellauf, 81
 - Daten, 51, 98
- morphologische Synthese, 13, 92

- Beispiellauf, 92
- Daten, 51, 98
- Nomen, 54
- Nomen des Deutschen, 52
- Nomen des Englischen, 53
- Nomenflexion, 14
- Parameter
 - `alias`, 108
 - `ap`, 113
 - `constr`, 109
 - `count`, 113
 - `default`, 107
 - `display`, 106
 - `edge`, 111
 - `extend`, 106
 - `fip`, 107
 - `hfc`, 114
 - `idtrace`, 109
 - `parser`, 110
 - `propose`, 112
 - `syntax`, 114
 - `time`, 107
 - allgemein, 106
 - Anapherninterpretation, 114
 - Parser, 107
 - Termersetzungsverfahren, 114
- Parameter `edge`
 - Beispiellauf, 111
- Parameter `fip`
 - Beispiellauf, 108
- Parameter `parser`
 - Beispiellauf, 110
- Parameter `propose`
 - Beispiellauf, 112
- Parser
 - Daten, 56, 98
 - MÜ-System-Parameter, 107
- Personalpronomen, 55
- Personalpronomen (Flexion), 13
- Possessivpronomen, 53
- Prädikat, 43
 - Ausschluß vom Backtracking, 73
 - Synonym, 72
- Produktionssystem, 17
- Programm, 35
- Prolog, 35
- PS, 17
- Quantoren, 54
- Reflexivpronomen, 55
- Relativpronomen, 53
- Repräsentation
 - ATL, 10
 - Ausgabe, 95
 - Löschen, 95
 - Speichern, 96
- semantische Analyse, 62, 83
 - Beispiellauf, 84
 - Daten, 60, 98
 - Laden, 101
- semantische Rolle, 8
- semantisches Merkmal, 9
- Speichern
 - von Repräsentationen, 96
- Speifikationsmerkmals, 59
- Struktur
 - Ausgabe, 74
- Substantivflexion, 14
- SUTRA-System
 - Kurzbeschreibung, 13
- Synonym
 - für Prädikat, 72
- syntaktische Analyse, 14, 81
 - Beispiellauf, 82
 - Daten, 98
 - Laden, 100
- System-Parameter
 - verändern, 105
- TBox-Konzept, 51
- TE-Verfahren, *siehe* Termersetzungsverfahren
- Term
 - Grund ..., 17
 - offen, 17
- Termersetzungssystem
 - Daten, 47
 - für die konzept. Analyse, 23
- Termersetzungsverfahren, 80
 - MÜ-System-Parameter, 114
- TES
 - Daten, 47
- Textrepräsentation
 - Laden, 103
 - referentiell, 10

- Tiefenkasus, 8
- Transfer, 65, 88
 - Beispiellauf, 89
 - Daten, 65, 99
 - Laden, 103
- Übergangslexikon FAS $\rightarrow$ TBox, 34, 51,
61, 65
- Übergangslexikon GPSG $\rightarrow$ FAS, 34, 51
- Übersetzung, 92
 - Beispiellauf, 93
- Unifikation
 - Implementation, 36
 - von Bäumen, 36
- Verb, 55, 56
- Verb des Deutschen, 52
- Verb des Englischen, 53
- Verbflexion, 13
- Verzweigungspunkte, 18
- Wissensrepräsentation, 1, 10